University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

PIOTR PRZYMUS

# QUERY OPTIMIZATION IN HETEROGENEOUS CPU/GPU ENVIRONMENT FOR TIME SERIES DATABASES

PhD dissertation

SUPERVISOR:

dr hab. Krzysztof Stencel
Institute of Informatics,
University of Warsaw

CO-SUPERVISOR:

dr Krzysztof Kaczmarski
Faculty of Mathematics and Information Science,
Warsaw University of Technology

Toruń, March 2014

Author's declaration: aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

Piotr Przymus,
19th March 2014

Supervisor's declaration: the dissertation is ready to be reviewed.

dr hab. Krzysztof Stencel,
19th March 2014

## ABSTRACT

In recent years, processing and exploration of time series has experienced a noticeable interest. Growing volumes of data and needs of efficient processing pushed the research in new directions, including hardware based solutions.

Graphics Processing Units (GPU) have significantly more applications than just rendering images. They are also used in general purpose computing to solve problems that can benefit from massive parallel processing. There are numerous reports confirming the effectiveness of GPU in science and industrial applications. However, there are several issues related with GPU usage as a databases coprocessor that must be considered.

First, all computations on the GPU are preceded by time consuming memory transfers. In this thesis we present a study on lossless lightweight compression algorithms in the context of GPU computations and time series database systems. We discuss the algorithms, their application and implementation details on GPU. We analyse their influence on the data processing efficiency, taking into account both the data transfer time and decompression time. Moreover, we propose a data adaptive compression planner based on those algorithms, which uses hierarchy of multiple compression algorithms in order to further reduce the data size.

Secondly, there are tasks that either hardly suit GPU or fit GPU only partially. This may be related to the size or type of the task. We elaborate on heterogeneous CPU/GPU computation environment and optimization method that seeks equilibrium between these two computation platforms. This method is based on heuristic search for bi-objective optimal execution plans. The underlying model mimics the commodity market, where devices are producers and queries are consumers. The value of resources of computing devices is controlled by supply-and-demand laws. Our model of the optimization criteria allows finding solutions for heterogeneous query processing problems where existing methods have been ineffective. Furthermore, it also offers lower time complexity and higher accuracy than other methods.

The dissertation also discusses an exemplary application of time series databases: the analysis of zebra mussel (*Dreissena polymorpha*) behaviour based on observations of the change of the gap between the valves, collected as a time series. We propose a new algorithm based on wavelets and kernel methods that detects relevant events in the collected data. This algorithm allows us to extract elementary behaviour events from the observations. Moreover, we propose an efficient framework for automatic classification to separate the control and stressful conditions. Since zebra mussels are well-known bioindicators this is an important step towards the creation of an advanced environmental biomonitoring system.

KEYWORDS: GPU, CUDA, time series, database, lightweight compression, multi-objective optimization, query optimization, economic models, wavelets, data mining

ACM COMPUTING CLASSIFICATION: Computing methodologies~Graphics processors, Information systems~Query optimization, Information systems~Data compression, Mathematics of computing~Time series analysis

# STRESZCZENIE

W ostatnich latach przetwarzanie i badanie szeregów czasowych zyskało spore zainteresowanie. Rosnące ilości danych i potrzeba ich sprawnego przetwarzania nadały nowe kierunki prowadzonym badaniom, które uwzględniają również wykorzystanie rozwiązań sprzętowych.

Procesory graficzne (GPU) mają znacznie więcej zastosowań niż tylko wyświetlanie obrazów. Coraz częściej są wykorzystywane przy rozwiązywaniu problemów obliczeniowych ogólnego zastosowania, które mogą spożytkować możliwości przetwarzania masywnie równoległego. Wiele źródeł potwierdza skuteczność GPU zarówno w nauce, jak i w zastosowaniach w przemyśle. Jest jednak kilka kwestii związanych z użyciem GPU jako koprocesora w bazach danych, które trzeba mieć na uwadze.

Po pierwsze, wszystkie obliczenia na GPU są poprzedzone czasochłonnym transferem danych. W pracy zaprezentowano rezultaty badań dotyczących lekkich i bezstratnych algorytmów kompresji w kontekście obliczeń GPU i systemów baz danych dla szeregów czasowych. Omówione zostały algorytmy, ich zastosowanie oraz szczegóły implementacyjne na GPU. Rozważono wpływ algorytmów na wydajność przetwarzania danych z uwzględnieniem czasu transferu i dekompresji danych. Ponadto, zaproponowany został adaptacyjny planer kompresji danych, który wykorzystuje różne algorytmy lekkiej kompresji w celu dalszego zmniejszenia rozmiaru skompresowanych danych.

Kolejnym problemem są zadania, które źle (lub tylko częściowo) wpisują się w architekturę GPU. Może być to związane z rozmiarem lub rodzajem zadania. W pracy zaproponowany został model heterogenicznych obliczeń na CPU/GPU. Przedstawiono metody optymalizacji, poszukujące równowagi między różnymi platformami obliczeniowymi. Opierają się one na heurystycznym poszukiwaniu planów wykonania uwzględniających wiele celów optymalizacyjnych. Model leżący u podstaw tego podejścia naśladuje rynki towarowe, gdzie urządzenia są traktowane jako producenci, konsumentami są natomiast plany zapytań. Wartość zasobów urządzeń komputerowych jest kontrolowana przez prawa popytu i podaży. Zastosowanie różnych kryteriów optymalizacji pozwala rozwiązać problemy z zakresu heterogenicznego przetwarzania zapytań, dla których dotychczasowe metody były nieskuteczne. Ponadto proponowane rozwiązania wyróżnia mniejsza złożoność czasowa i lepsza dokładność.

W rozprawie omówiono przykładowe zastosowanie baz danych szeregów czasowych: analizę zachowań racicznicy zmiennej (*Dreissena polymorpha*) opartą na obserwacji rozchyleń muszli zapisanej w postaci szeregów czasowych. Proponowany jest nowy algorytm oparty na falkach i funkcjach jądrowych (ang. kernel functions), który wykrywa odpowiednie zdarzenia w zebranych danych. Algorytm ten pozwala wyodrębnić zdarzenia elementarne z zapisanych obserwacji. Ponadto proponowany jest zarys systemu do automatycznego oddzielenia pomiarów kontrolnych i tych dokonanych w stresujących warunkach. Jako że małże z gatunku *Dreissena polymorpha* są znanymi wskaźnikami biologicznymi, jest to istotny krok w kierunku biologicznych systemów wczesnego ostrzegania.

SŁOWA KLUCZOWE: GPU, CUDA, szeregi czasowe, bazy danych, lekka kompresja, wielocelowa optymalizacja, optymalizacja zapytań, modele ekonomiczne, falki, eksploracja danych

# PUBLICATIONS

Publications included in this dissertation:

[58] Przymus, P. and Kaczmarski, K. Improving efficiency of data intensive applications on GPU using lightweight compression. In *On the Move to Meaningful Internet Systems: OTM 2012 Workshops, Lecture Notes in Computer Science*, volume 7567, pages 3–12. Springer Berlin Heidelberg, 2012.

[59] Przymus, P. and Kaczmarski, K. Time series queries processing with GPU support. In *New Trends in Databases and Information Systems*. 17th East-European Conference on Advances in Databases and Information Systems September 1–4, 2013, Genoa, Italy, 2013.

[60] Przymus, P. and Kaczmarski, K. Dynamic compression strategy for time series database using GPU. In *New Trends in Databases and Information Systems*. 17th East-European Conference on Advances in Databases and Information Systems September 1–4, 2013, Genoa, Italy, 2013.

[61] Przymus, P., Rykaczewski, K., and Wiśniewski, R. Application of wavelets and kernel methods to detection and extraction of behaviours of freshwater mussels. *Future Generation Information Technology*, pages 43–54, 2011.

[62] Przymus, P., Rykaczewski, K., and Wiśniewski, R. Zebra mussels' behaviour detection, extraction and classification using wavelets and kernel methods. *Future Generation Computer Systems*, 33(0):81–89, 2014. Special Section on Applications of Intelligent Data and Knowledge Processing Technologies. Guest Editor: Dominik Ślęzak.

[63] Przymus, P., Kaczmarski, K., and Stencel, K. A bi-objective optimization framework for heterogeneous CPU/GPU query plans. In *CS&P 2013 Concurrency, Specification and Programming*. Proceedings of the 22nd International Workshop on Concurrency, Specification and Programming, September 25–27, 2013, Warsaw, Poland.

Publications included in this dissertation are divided into two parts: query optimization in time series databases and applications. Results associated with the query optimization in time series databases were described in the publications [58–60, 63]. Applications have appeared previously in the publications [61, 62].

It is worth noting that two of the published publications aroused more interest: Firstly, the authors received *Best paper award* on *Future Generation Information Technology* 2012 conference for publication [61] and were thus invited to prepare an extended version of this paper which appeared in *Future Generation Computer Systems* journal [62]. Secondly, publication [60] was selected as one of best papers of *GPUs In Databases 2013* workshop, a satellite event of *17th East European Conference on Advances in Databases and Information Systems*.

# ACKNOWLEDGEMENTS

I want to thank my supervisors, dr Krzysztof Kaczmarski and dr hab. Krzysztof Stencel, for their encouragement, patience and support without which this thesis would not have been possible.

I would like to thank my wife and family for their love and support during all these years it took me to complete this thesis. I would also like to thank my friends, who have made my graduate years so enjoyable.

I acknowledge, with gratitude, the generous financial support which helped me to write this thesis:

- The support from the Marshall of Kuyavian-Pomeranian Voivodeship in Poland with the funds from European Social Fund (EFS) (a part of integrated operational program for regional development, activity 2.6) in the form of a grant for PhD students (Step in the future program, second edition and fifth edition).
- The support from Nicolaus Copernicus University, grant 1748-M (2013).
- The support from National Science Centre (NCN), grant decision number DEC-2012/07/D/ST6/02483, that funded parts of the research presented as well as the future one.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LISTINGS

# ACRONYMS

| | |
|---|---|
| CPU | Central Processing Unit |
| CUDA | Compute Unified Device Architecture |
| FPGA | Field Programmable Gate Array |
| GPU | Graphical Processing Unit |
| GPGPU | General–Purpose computing on Graphics Processor Units |
| OpenCL | Open Computing Language |
| PCI | Peripheral Component Interconnect |
| PCIe | Peripheral Component Interconnect Express |
| RAM | Random Access Memory |
| SCADA | Supervisory Control And Data Acquisition |
| SIMD | Single Instruction, Multiple Data |
| TSDS | Time Series Database System |

1

# INTRODUCTION

## Contents

## 1.1 MOTIVATION

In recent years, there has been an increased interest in processing and exploration of time series data. There are many ongoing studies in this field of time series analysis and data mining [24, 32, 40, 45, 58, 64, 65]. Consequently, this raises interest in specialized databases systems for time series [21, 34, 45, 55, 58–60, 65], which could support advanced analysis and data mining methods.

Such database systems play an important role in many industrial and scientific systems like Supervisory Control And Data Acquisition (SCADA), servers and services monitoring, measurements acquisition of scientific experiments or stock market data procurement [21, 46, 49, 61, 62]. These systems are expected to process and store millions of data points per minute, 24 hours a day, seven days a week, generating terabytes of logs. Due to checking of regression errors and early malfunction prediction, such data must be kept with proper resolution and include all details. The number of time series generated in this way grows very quickly and falls under the category of *Big Data* (i. e. problems in which the size of data is a problem itself).

Traditional solutions such as relational databases supported with specialized tools show their limitations when handling a large number of time series [21]. Although relational database management systems can be adapted to store time series using base types (e. g. by storing one row per timestamped data entry or by storing multiple values as a binary large object), performance of such solutions is usually poor and inefficient [34, 81]. This is mostly due to the fact that most of the storage and processing systems were not designed for time series data. Moreover, because of limited support for time series processing, database systems must be accompanied by external tools, like R [73], RRDtool [54], SAS [1] and others, which further reduces the performance of the whole system. On the other hand, these tools may be used as stand-alone systems for time series processing. Unfortunately, usually they are optimized for medium datasets and thus are not capable of handling large datasets.

For this reasons, extensive studies are conducted in order to find new methods and effective solutions. It is worth to mention research on: new algorithms and representation models [22, 41, 45, 47, 65], hardware solutions [15, 36, 59, 60, 68, 78], specialized extensions of relational databases [34] and processing software [75], or usage of non-relational databases [19, 31, 37, 50, 55, 74].

In this study, we investigate a mixed approach by utilizing hardware solutions (like GP-GPU, General-Purpose computing on Graphics Processing Units) together with Central Processing Unit (CPU) computing (heterogeneous query planner), exploring new representation methods and algorithms (lightweight compression planner and GPU query processing) or utilizing NoSQL databases.

General-purpose computing on graphics processing units involves the use of graphics processing units (GPU) in tasks traditionally handled by central processing units. GPUs offer a notable processing power for tasks that can benefit from massive parallel processing.

Execution of database queries is an example of a successful application of GPGPU. The current research focuses on using the GPU as a co-processor [15, 59, 60, 78]. GPU as a co-processor may accelerate numerous database computations, e. g. relational query processing, query optimization, database compression, or supporting time series databases [15, 59, 60].

An application of GPU requires transferring data from the CPU memory to the graphical device memory. This data transfer is usually time-consuming. It may diminish the gain of the acceleration credited to GPU.

One of the possibilities, and often the only option, to optimize the mentioned data transfer is to reduce its size by data compression. Classical compression algorithms are computationally expensive (gain from the data transfer does not compensate the calculations [83]) and difficult to implement on the GPU [80]. Alternatives such as lightweight compression algorithms, which are successfully used for CPU and GPU, are therefore very attractive [3, 9, 25, 58–60, 82, 83].

However, lightweight compression does not solve all the problems. In particular, GPU is optimized for numerical computation and thus, only selected operations will benefit from GPU. Small datasets are another problem. For such sets, the data transfer may dominate processing time and destroy the performance gain. Therefore, joint processing capabilities of both CPU and GPU are worth considering. Furthermore, as it is common to have more than one GPU in a computer, a potential use of various GPU devices should be considered. This type of query plans is called heterogeneous.

The previous research efforts focused on the creation of query plans based on a cost model. This approach finds plans with the best throughput. However, it does not allow modelling all phenomena that can occur in heterogeneous systems. Performing a query as soon as possible is not always cost efficient [27]. For this reason, we propose a query processing model based on concepts of markets that are known to be suitable for describing the interactions in a heterogeneous world.

## 1.2 PROBLEM STATEMENT

The subject of this PhD dissertation is query optimization in heterogeneous CPU/GPU environment for time series databases. The main objectives were the following:

- to optimize data transfer and storage size, by proposing a lightweight compression framework and a compression planner [58–60],
- to optimize query processing in heterogeneous CPU/GPU environment through development of bi-objective optimization framework [63],
- to develop a prototype CPU/GPU query processing module [58, 59].

In addition, possible applications of such time series databases are discussed, in particular:

- a biomonitoring system [61, 62],
- and a network monitoring system [59, 60].

## 1.3 DISSERTATION OUTLINE

The dissertation consists of three introductory chapters, six publications placed in an appendix, and description of experiments also placed in an appendix.

We present the necessary preliminaries in Chapter 2. In particular, in Section 2.1, Section 2.2 and Section 2.3, we discuss time series, time series databases, time series data mining and query processing, respectively. Lightweight compression methods are discussed in Section 2.4. A short introduction to general-purpose computing on graphics processing units is provided in Section 2.5. In Section 2.6, general terms of economic models for resource management and scheduling in computational environments are presented. Final section of this chapter, Section 2.7, introduces basic concept of multi-objective optimization.

In Chapter 3, we summarize contribution of this dissertation, that is, publications gathered in Appendix A. Appendix B provides description of experiments conducted in this dissertation.

# 2 PRELIMINARIES

## Contents

## 2.1 TIME SERIES

Measurements performed over time play an important role in many scientific fields. Data sets collected in such a way are called time series.

**Definition 1.** A *time series* is a sequence of observations, measured at successive points in time. We will denote a time series by $V = \{(t, v_t) \colon t \in T\}$, where $T$ is the time index set of this series. If $V$ is finite and has $n \geqslant 1$ elements, we will say that $V$ is a *time series of length* $n$.

There are many applications where time series data is gathered and analysed. This involves a variety of data sources:

- stock price and trading volumes, foreign exchange (Forex) quotes,
- smart meters measurements (e. g. electricity meters) or industrial supervisory control and data acquisition (SCADA) systems,
- medical records like electrocardiography (ECG) recordings,
- servers, services and network performance records,
- scientific experiments measurements (seismogram, biomonitoring).

Chart plots are the most obvious way to start time series analysis. Example of such a plot is presented in Figure 2.1.

There is a number of characteristics which can be used to classify time series data. The first of them is associated with the measurement time. Depending on the characteristic of time index $T$, we distinguish two types of time series data: evenly and unevenly spaced.

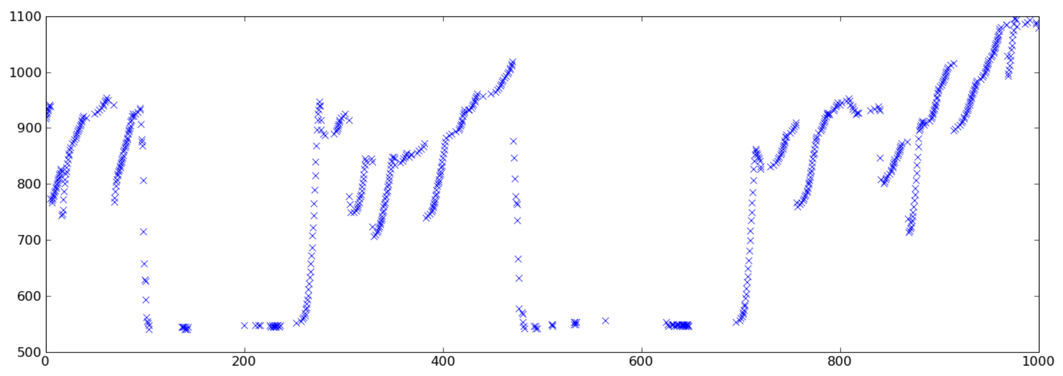


Figure 2.1: Time series example – unevenly spaced measures of mussel activity from biomonitoring system described in [61, 62].

**Definition 2.** An *evenly spaced* (*unevenly spaced*) time series is a time series with constant (irregular) length of intervals between consecutive observations.

In many applications, it is safer to assume that gathered time series are unevenly spaced[1], because often large systems cannot guarantee either constant measurement period or correct measurement and data transfer.

The next characteristic is associated with the type of the measured variable. A time series can be classified as being either a stock (level variable) or a flow (rate variable) series, depending on the type of measurements being taken [2, 4]. This distinction turns out to be important in some processing tasks like interpolation of missing values. A *stock variable* is measured at one specific time $t$ and represents a quantity existing at that point in time. A *flow variable* measures the flow of changes in the stock, i. e. it is a derivative of a stock variable.

Another distinction is associated with the number of observed variables, time series may by either univariate or multivariate.

**Definition 3.** A time series is said to be:

- a *univariate time* series if it consists of single variable observations,
- a *multivariate time* series if observations span over multiple variables within the same time range.

From the point of view of statistics, a time series is a realization of a discrete stochastic process. An extensive course of statistical time series analysis may be found in [71].

**Definition 4.** Let $T \subset [0, \infty)$. A family of random variables $\{X_t : t \in T\}$, indexed by $T$, is called a *stochastic process*. If $T = \mathbb{N}$, $\{X_t : t \in T\}$ is said to be a discrete-time process, and when $T = [0, \infty)$, it is called a continuous-time process.

1 Note that a time series with equal spacing with some missing data is naturally treated as unevenly spaced.

Depending on the nature of stochastic process behind a time series, we may distinguish stationary or non stationary cases.

**Definition 5.** A stochastic process $\{X_n : n \geqslant 0\}$ is said to be *stationary* if the random vectors $(X_0, X_1, \dots, X_k)$ and $(X_m, X_{m+1}, \dots, X_{m+k})$ have the same joint distribution for all $m, k \geqslant 0$.

Since the above definition is too strict for practical applications, it is useful to introduce one more notion, weaker than stationarity.

**Definition 6.** A stochastic process $X_n : n \leqslant 0$ is said to be *weakly stationary* if the following holds:

- $\mathbb{E}(X_n^2) \leqslant \infty$ for all $n \geqslant 0$,
- $\mathbb{E}X_n$ is the same for all $n \geqslant 0$,
- $Cov(X_m, X_n) := \mathbb{E}((X_m - \mathbb{E}X_m)(X_n - \mathbb{E}X_n)$ depends only on $n - m$

($\mathbb{E}(\cdot)$ stands for the expected value).

Less formally, it means that a stochastic process is stationary if there is no systematic trend, no systematic change in variance, and if strictly periodic variations or seasonality do not exist. Most processes in nature appear to be non-stationary [71]. However, there is a broad field of theory that is only applicable to stationary processes.

A common approach in time series analysis is frequency domain analysis. Fourier transform can be used to analyse the frequency spectrum. However, it does not provide any information when a frequency component is present. Wavelets, are examples of Multiscale Resolution Analysis, meaning that wavelet coefficients contains the information about the frequency and the time domain. Wavelets may also be used in construction of signal filters. A *wavelet filter* is a non-linear digital filtering technique necessary to perform a high degree noise reduction in a signal. A broad introduction to wavelets theory may be found in [18]. See [62] in Appendix A.6 on page 81 for a short introduction to necessary concepts.

Another common tool used in time series analysis is known as kernel smoother. A *kernel smoother* is a statistical technique for estimating values of a function, by using its noisy observations. It is mostly used for smoothing the gathered observations.

In the context of databases and data mining it is convenient to define the term of a time series subsequence [24]:

**Definition 7.** Given a time series $V = \{(t_i, v_{t_i}) : 1 \leqslant i \leqslant n\}$ with time index set $T_V = \{t_1, \dots, t_n\}$, a *subsequence* $W$ of $V$ is a time series of length $m \leqslant n$, consisting of values corresponding to $m$ contiguous time instants from $V$:

$$W = \{(t_i, v_{t_i}) : k \leqslant i \leqslant k + m - 1\}$$

for some $1 \leqslant k \leqslant n - m + 1$. In particular, the time index set of $W$ is of the form $T_W = \{t_k, \dots, t_{k+m-1}\} \subset T_V$. We denote the set of all subsequences of length $m$ of $V$ as $S_V^m$.

## 2.2 TIME SERIES DATABASES

A *Time Series Database System (TSDS)* is a database system optimized towards storing, processing and querying of large time series collections.

**Definition 8.** A *time series database* DB is an unordered set of time series.

Usually a time series database system supports defining the structure of the data, control of data storage, facilities supporting data load (bulk or individual). Data is then retrieved and processed using a query engine and processing tools. Often results are presented in a graphical form.

Example

**TSDS queries**

- Find the server with highest processor load during the (last) weekend.
- Compute daily average processor temperature in the last month.
- Find companies with similar stock prices over a time interval.
- Find products with similar sell cycles (stock levels over time analysis).
- Cluster users with similar credit card utilization (expenses in time).

TSDS usually stores time series data in a special format within the underlying database storage system. There are successful examples of building a TSDS on top of existing relational [34, 69, 81] or non-relational database systems [37, 55, 74]. There exist systems with dedicated storage infrastructure [21, 54] as well.

Furthermore, the database should be able to store additional metadata used to describe the stored time series. For instance, in OpenTSDB it is assumed that each time series is identified by a *metric name* and may be additionally marked with a set of *tags* describing measurement details.

Example

**Metadata associated with time series**

metric: http.hits - hits to apache
tags: host=A webserver=static

Many systems also support the process of data collecting from different sources by providing data load facilities like distributed data collectors (often with limited data processing capabilities), which may work directly on data sources and push data into the data storage.

Sometimes a TSDS supports time series compaction, i. e. a process which splits a time series into several parts, defined by packing subsequences. Often these packing subsequences are just time intervals of a specified length (e. g. 15 minutes, 1 hour, 24 hours – depending on the number of observations). Formally, disjoint time series subsequences are stored in separate records.

## 2.3 time series data mining and query processing

As there is no official query and processing standard for TSDS, capabilities of existing systems vary, depending on the implementation. However, some basic functionality for querying and data transformations is similar among many TSDS. Usually there is support for:

- selective query using metadata and time criteria;
- basic time series transformations like: aggregation[2], downsampling[3] and interpolation;
- conversion between evenly and unevenly sampled time series;
- graphical presentation of the results.

The most basic form of querying time series database is done by specifying metadata and time criteria.

**Definition 9.** A *metadata selective query* is a query which selects time series according to the given metadata criteria. For a time series database DB and metadata criteria $C_{meta}$, such a query returns the following set:

$$\{V \in DB : \text{ metadata of } V \text{ match criteria } C_{meta}\}.$$

Example

**Metadata selective query**

metric: http.hits
tags: host=* webserver={static, dynamic}

2 E. g. minimum, maximum, average, sum. See [55] for details.
3 Reduction of the sampling rate, see [55].

**Definition 10.** A *time selective query* is a query which for each given time series returns the largest possible subsequence, which matches the given time criteria. For a subset $A$ of a time series database $DB$ and time criteria $C_{time}$ (i. e. some time interval), such a query returns the following set:

$$\{V|_{C_{time}} : V \in A\}, \text{ where } V|_{C_{time}} := \{(t, v_t) : t \in T_V \cap C_{time}\} \text{ for } V = \{(t, v_t) : t \in T_V\}.$$

Those types of queries may be combined into a selective query.

Example

**Selective query**

metric: http.hits
tags: host=* webserver={static, dynamic}
from: 2012-10-10 to: 2012-12-10

Selected data is usually subjected to data transformations. Depending on system implementation, this may vary, but commonly interpolation, aggregation and downsampling may be found:

- interpolation is necessary when working with unevenly spaced time series,
- aggregation results in lossy dimensionality reduction (see below),
- downsampling is used to reduce the sampling rate.

Aggregation works on equally sampled time series. For each time point aggregation across the data values in all time series is calculated. If the aggregated time series are not equally sampled interpolation is used.

The following definitions are commonly used in the field of time series data mining and data bases, see, e. g. , [24].

Many time series data mining methods rely on the notion of similarity between time series.

**Definition 11.** A *similarity measure* $D$ is a function taking two time series $V_1$ and $V_2$ as inputs. The returned value $D(V_1, V_2) \geqslant 0$ is interpreted as the distance between the time series $V_1$ and $V_2$. If $D$ additionally satisfies the symmetry condition $D(V_1, V_2) = D(V_2, V_1)$ and the triangle inequality $D(V_1, V_3) \leqslant D(V_1, V_2) + D(V_2, V_3)$, it is called a *metric*.

It is also convenient to define the notion of a subsequence similarity measure. It is used to represent the distance between one time series $V_1$ and its best matching location in another time series $V_2$.

**Definition 12.** For time series $V_1, V_2$ with $|V_1| < |V_2|$, the *subsequence similarity measure* is defined as $D_{seq}(V_1, V_2) := \min\{D(V_1, V) : V \in S_{V_2}^{|V_1|}\}$.

There is a whole number of works on defining similarity measures using different techniques. In some applications metrics like Euclidean or Hamming distance are sufficient [24]. Typically, these are low computationally demanding methods.

In other applications more advanced methods are used. For example, *dynamic time warping (DTW)* is an extremely popular algorithm [39, 40, 42, 43] for measuring similarity between two time series which may vary in time or speed. The main idea behind this algorithm is to perform elastic transformation of time series in order to detect similar shapes with different phases.

It is also worth to mention methods which rely on the computation of a feature set reflecting various aspects of the series. There are known examples of using similarity measures based on fuzzy sets [7, 33, 77] or wavelets [5, 35, 44, 66, 76]. For an extensive list of used methods see [24].

Having defined the concept of a similarity measure, we can expand the notion of TSDS queries.

**Definition 13.** Consider a time series database DB. Let Q be a query time series and D be a similarity measure. Moreover, let $K \geqslant 1$ and $\epsilon > 0$. Then:

- *a query by content* is a query which returns an ordered list $\{V_1, \ldots, V_n\}$ of time series from the database DB (or from its given subset) such that $D(Q, V_i) \leqslant D(Q, V_j)$ for all $1 \leqslant i < j \leqslant n$;

- *a* K*-nearest neighbours query* is a query which returns a set of K series from DB which are the most similar to Q. More precisely, it returns $A \subseteq DB$ such that $|A| = K$ and for all $V \notin A$, $D(Q, V) \geqslant \min_{W \in A} D(Q, W)$;

- *an* $\epsilon$*-range query (a whole series/pattern matching query)* is a query which returns the set of all series from DB (or from its given subset) which are within distance at most $\epsilon$ from Q, i. e. $\{V \in DB : D_{seq}(Q, V) \leqslant \epsilon\}$;

- *an* $\epsilon$*-range query (a subsequence matching query)* is a query which returns all subsequences $W$ of all series $V \in DB$ such that $D_{seq}(Q, W) \leqslant \epsilon$, i. e. $\{W \in \bigcup_{m \geqslant 1} S_V^m : V \in DB, D_{seq}(Q, W) \leqslant \epsilon\}$.

In many applications, the problem of classification of time series is considered. This applies both to time series and their subsequences. Below we recall the basic concepts associated with classification (see also [62] in Appendix A.6 on page 81). For an extensive study on classification problems see [6, 23]

*Classification* is the problem of identifying categories to which a new observation belongs. A training set of observations with known category membership is used for adjusting model parameters.

Data dimensionality reduction is an important aspect of data mining and is mainly used to avoid the curse of dimensionality[4]. *Feature extraction* is a form of dimensionality reduction, where the input data is transformed into a reduced representation set of features.

An important aspect of classification is the validation of the results. One of the most popular approaches is a $m$*-fold cross-validation*. It is a model validation technique where data set is randomly divided into $m$ disjoint equal size sets. The model is trained $m$ times, each time with a different set held out as training set (a subset used for adjusting the model parameters) and complement validation set (used to estimate the generalization error). The estimated performance is the mean of $m$ results.

## 2.4 LIGHTWEIGHT COMPRESSION

Compression is a process of reducing the amount of storage needed to represent a certain set of information. Compression rate (compression ratio) is the achieved reduction in memory usage. Compression aims to minimize the amount of data that need to be held, handled, and/or transmitted by a computer. Most commonly a compression algorithm identifies and eliminates statistical redundancy in processed data to increase the efficiency of memory consumption. Other methods involve discarding certain bits of information.

Compression algorithms can be distinguished by the manner in which the data is being reconstructed and can be either lossless or lossy. *Lossless compression algorithm* is a data encoding method that allows the original data to be perfectly reconstructed from the compressed data. No information is lost in lossless compression. *Lossy compression algorithm* is a data encoding method that compresses data by discarding (losing) some of it.

In the context of time series data, lossy compression algorithms often approximate data using splines, piecewise linear approximation or extrema extraction [26]. By simplifying the data representation, one can achieve good compression ratio and decompression speed. Unfortunately, this is done at the cost of data anomalies degradation, so lossy compression algorithms cannot be used in applications where anomalies analysis matters.

Another distinction of compression algorithms is based on the main optimization objective, i. e. compression ratio or compression/decompression speed. Compression algorithms differ in the amount of time that is required to compress and decompress data, as well as the achieved compression rate. Some compression algorithms, like Lempel–Ziv or Burrows–Wheeler transform, perform complex analysis of the data to achieve the highest possible compression rate, usually on the cost of increased run-time.

On the other side, there is a family of *lightweight compression algorithms*, primarily intended for real-time applications, which favours compression/decompression speed over compression ratio.

Compression has been long considered as an effective mean to reduce the data footprint in databases, especially for column-oriented databases [3, 8, 9, 17, 20, 25, 58–60, 82, 83]. This is an effective method to improve overall query processing performance by reducing the data transfer time. In disk-based databases, disk accesses are far slower than the decompression throughput on the CPU, so it pays off to reduce the data access time at the cost of more computation on the CPU.

In particular, usage of lightweight compression algorithms has been extensively studied in the context of databases. With these compression schemes, most queries can be evalu-

4 Miscellaneous phenomena that appear when analysing data in high-dimensional spaces that do not occur in low-dimensional cases.

ated after partial decompression or decompression on the fly. Additionally, in some cases query may be processed without any decompression. Furthermore, lightweight compression can easily be vectorized which is an important advantage. Examples of lightweight compression algorithms may be found in [60] (see Appendix A.2 on page 37).

The main drawback of many lightweight compression schemes is that they are prone to outliers in the data frame. *Outliers* are observations points that are distant from the other observations in the sample. Determining if an observation is an outlier strongly depends on use case. For example, consider the following data frame: $\{1, 2, 3, 2, 2, 3, 1, 1, 64, 2, 3, 1, 1\}$. The value 64 can significantly affect the resulting compression ratio for some lightweight compression algorithms. A common solution is a *PATCH* modification of the lightweight compression algorithms which stores outliers separately (see publication [60] in Appendix A.2 on page 37).

In some cases, it is worth to investigate a scenario where multiple lightweight compression schemes are used together and form a compression plan (the output from one algorithm becomes the input for the next). *Cascaded lightweight compression plan* is a compression scheme which uses hierarchy of multiple lightweight compression algorithms in order to further reduce the data size.

Given a number of individual compression algorithms and a data set the main aim of a compression planner is to return a feasible and good combination of the individual compression schemes. This approach often improves the achieved compression ratio but is more computationally intensive. Therefore, it is crucial to maintain a good balance between compression ratio and compression/decompression speed [25, 60].

## 2.5 general-purpose computing on graphics processing units

Originally GPU cards were designed for graphics rendering tasks. However, now they have evolved into massively multi-threaded many-core general-purpose computing coprocessors. Internally, the GPU consists of many SIMD (Single Instruction, Multiple Data) multiprocessors which share a piece of the GPU device memory. General overview of the GPU architecture may be found in Figure 2.2.

Currently there are two competing general-purpose GPU computing languages, OpenCL and CUDA. CUDA is a proprietary framework for NVIDIA GPUs, whereas OpenCL is an open standard which is supported by several GPU manufacturers (including NVIDIA, AMD and Intel). At the moment NVIDIA CUDA framework seems to be more popular and achieves better performance than OpenCL [38]. Although, in the future this may obviously change.

The GPU card has its own memory or a separate area in the CPU main memory. Thus, the data has to be explicitly transferred from the CPU main memory to the GPU main memory. Similarly, the results produced by the GPU have to be transferred back to the CPU main memory. This data transfer often introduces significant overhead. Hence, it is necessary to include the transfer cost in the total execution time of the operation [10–15, 58–60].

CUDA framework exposes the hierarchy of the GPU threads. The basic unit of execution in CUDA is a thread. Each thread executes the same function code. A function code that executes on the device is usually called a *kernel*. Threads are organized into blocks which are themselves organized into a grid. Threads in the same thread block share resources on one multiprocessor, e. g. registers and shared memory. Within a thread block, threads are

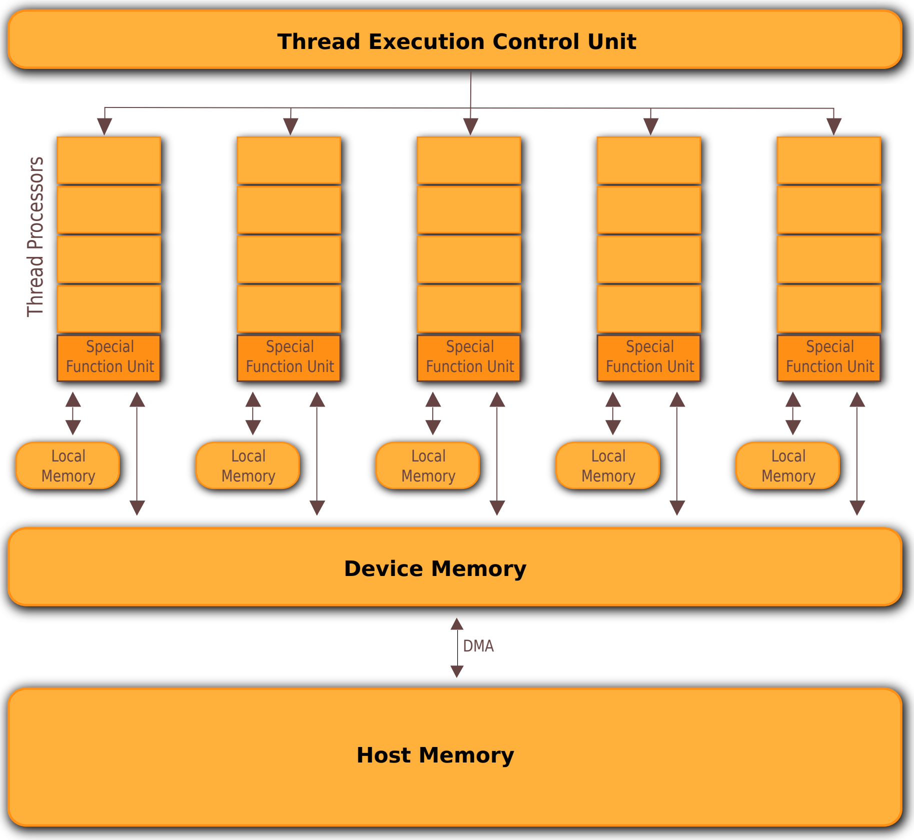


Figure 2.2: The NVIDIA GPU architecture

grouped into warps: *warps* are groups of 32 threads. Warps are scheduled across multiprocessors. Instructions are issued per warp.

See Listing 2.5.1 for CUDA code example. In the example sum of two vectors of size N is presented. In lines 2–7 a kernel function is defined, which will calculate the sum. Explicit memory copy to and from device are done in lines 20–21 and 26, respectively. Kernel execution is done in line 23, in this place also the grid and block size for kernel is defined. Additionally, see Figure 2.3 for an general overview of interaction between host and device as well as grid, threads and blocks organisation.

One of the major problems in the design of algorithms in SIMD architecture is the use of conditional statements. For example in CUDA framework, a warp executes one common instruction at a time. Threads in a warp may diverge via a data dependent conditional branch. Then the warp serially executes each branch path taken, disabling threads that are not on that path. Therefore, full efficiency is gained when all threads in a warp agree on their execution path [52, 53, 67].

CUDA also exposes the memory hierarchy to developers. An overview of different memory types available is presented in Table 2.1. As the performance of used memory type varies considerably, e. g. the global memory is hundreds of times slower than the shared one, proper use of memory is often a key aspect for highly efficient algorithms.

A GPU as a co-processor may accelerate numerous database tasks, e. g. query processing, query optimization and various other tasks [14, 78]. However, it is important to emphasize that a CPU accompanied by a GPU co-processor is a *shared nothing architecture* and data transfer usually is a significant issue.

Listing 2.5.1: Example CUDA code: sum of two vectors

```
#define N 10
__global__ void VectorAdd(int *a, int *b, int *c) {
    int tid = blockIdx.x; // handle the data at this index

    if (tid < N)
        c[tid] = a[tid] + b[tid];
}

int main( void) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;

    // allocate the memory on the GPU
    cudaMalloc((void **) &dev_a, N * sizeof (int));
    cudaMalloc((void **) &dev_b, N * sizeof (int));
    cudaMalloc((void **) &dev_c, N * sizeof (int));
    // Here you should fill the arrays ‘a’ and ‘b’ on the CPU

    // copy the arrays ‘a’ and ‘b’ to the GPU
    cudaMemcpy(dev_a, a, N * sizeof (int), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, N * sizeof (int), cudaMemcpyHostToDevice);

    VectorAdd <<<N,1>>> (dev_a, dev_b, dev_c);

    // copy the array ‘c’ back from the GPU to the CPU
    cudaMemcpy(c, dev_c, N * sizeof (int), cudaMemcpyDeviceToHost);

    // Here you should cleanup the memory allocated on the GPU
    return 0;
}
```

Another interesting aspect is a query processing model which allows the use of various devices in processing. Contemporary computer systems often include more than one processing device like multi-core CPUs, multiple GPUs or Field Programmable Gate Array (FPGA) modules[5]. Therefore, an interesting problem arises whether these devices may co-operate when processing a query.

*Heterogeneous query processing* is a query processing model which combines multiple computational (CPUs and GPUs) units in a single query plan. The main problem of heterogeneous query processing is the construction of such a query plan that uses only computational units from which query performance will benefit most and yet will minimize used resources. Each device may have a different communication cost (e. g. PCIe — Peripheral Component Interconnect Express expansion bus or shared memory) with the CPU main memory. Furthermore, devices can often communicate directly between each other. Therefore, it is important to take these costs into consideration. See publication [63] in Appendix A.4, on page 55, for more details.

5 There have been recent studies on the use of FPGA in databases [51].

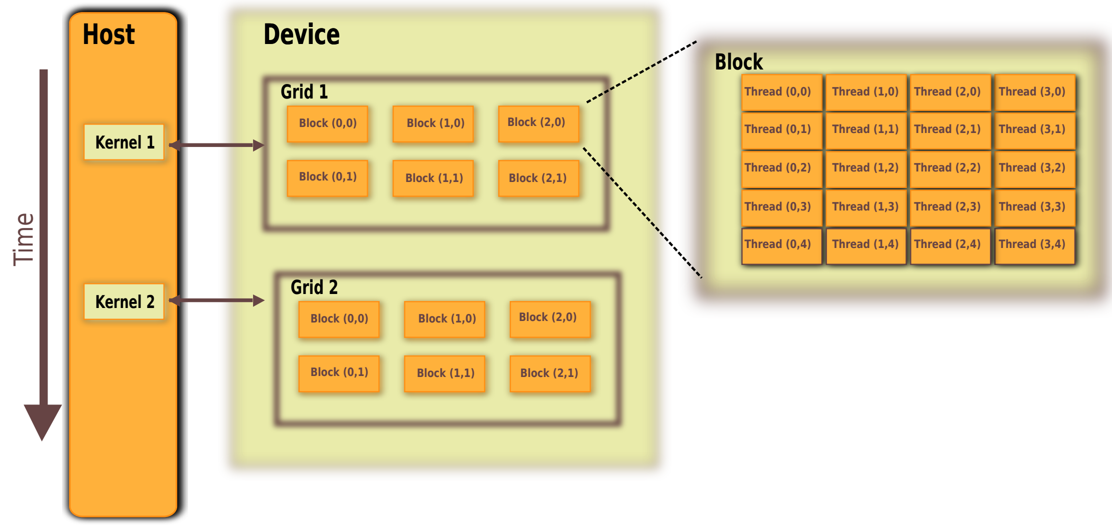


Figure 2.3: CUDA kernel run workflow

## 2.6 ECONOMIC MODELS FOR RESOURCE MANAGEMENT AND SCHEDULING

Extensive study on the subject of economic models in computing environments may be found in [16].

Economic models are known to be suitable for describing the interactions in heterogeneous environments. They have already gained a considerable interest in the context of computing in heterogeneous grid systems [16, 70], where they were used for resource management and scheduling. This concept has also reached distribution databases systems research, which resulted in several prototypes [16, 56, 72].

One of the most basic economic models is *commodity market* or *supply and demand driven pricing model* (see [16] for details). Supply and demand pricing is an economic model of price determination in a competitive market. The price for a particular good will vary until it settles at a point where the quantity demanded by consumers equals the quantity supplied by producers, resulting in an economic equilibrium for price and quantity.

In such a market, resource owners (processing devices) price their assets and charge their customers (queries) for consumed resources. The prices are being changed until equilibrium between supply and demand is found. Typically, the value of a resource is influenced by its strength, physical cost, service overhead, demand and preferences [16]. A consumer may be charged for various resources like CPU cycles, memory used, the bus usage or the network usage. Typically, a broker mediates between the resource owners and the consumer. The resource owners announce their valuation and the resource quality information (e. g. estimated time) in response to the broker's enquiry.

## 2.7 MULTI-OBJECTIVE OPTIMIZATION

Basic concepts of multi-objective optimization are included for the sake of completeness. An extensive survey on this subject may be found in [48]. We will be interested in the following problem:

$$\text{optimize } F(x) = \big(F_1(x), F_2(x), \ldots, F_k(x)\big),$$

| **Memory type** | **Mode** | **Description** |
|---|---|---|
| Registers | read/write per-thread | fastest but has only thread scope, limited |
| Shared | read/write per-block | limited, fast, but subject to bank conflicts permits exchange of data between threads in block |
| Constant | read/only per-grid | this is where constants and kernel arguments are stored, slow, but with cache |
| Texture | read/only per-grid | cache optimized for 2D access pattern |
| Global | read/write per-grid | slow, requires sequential and aligned read/writes to be fast (coalesced read/write), depending device capability may be cached or not cached |
| Local | read/write per-thread | used for whatever does not fit into registers, part of global memory, automatic coalesced reads and writes, slow, depending device capability may be cached or not cached |

Table 2.1: Memory types in CUDA framework – from fastest to slowest

where $k \geqslant 1$ is the number of objective functions, $x \in \mathbb{R}^n$ is a vector of design variables, $n$ is the number of independent variables $x_i$, and $F(x) \in \mathbb{R}^k$ is a vector given by the values of objective functions $F_i \colon \mathbb{R}^n \to \mathbb{R}$.

Typically, there is no single global solution, thus, a definition of an optimal solution set should be established. The mainly used approach in defining optimal solution is known as Pareto optimality.

**Definition 14.** A point $x^* \in R^n$ is called *Pareto optimal* if there does not exist another point $x \in R^n$ such that $F(x) \leqslant F(x^*)$ and $F_i(x) < F_i(x^*)$ for at least one function.

Note that, for some problems, there may be an infinite number of Pareto optimal points. Given a set of choices and a way of valuing them, the *Pareto set* consists of choices that are Pareto optimal.

In many computational problems, it is sufficient to find the most preferred Pareto optimal solution according to subjective preferences. Preferences are usually given by a decision maker. This problem may be solved using different approaches. It is worth to mention two of them, a priori methods and methods with no articulation of preferences. Examples of such methods may be found in [48]. In *a priori methods* the preferences are specified at the beginning by the decision maker and are used to find the best solution. In *methods with no articulation of preferences* a neutral compromise solution is identified without preference information.

A special case of multi-objective decision making is a *bi-objective optimization*, where optimal decisions need to be taken in the presence of trade-offs between two conflicting objectives.

The concept of multi-objective optimization has application in many engineering and computational problems [48]. One of the them is multi-objective query plan optimization [56, 72]. As this problem is NP-hard (reduction from knapsack problem), it is often sufficient to propose an approximate solution [56, 72].

# 3 DESCRIPTION OF THE CONTRIBUTION

## Contents

## 3.1 NOTES ON ARCHITECTURE FOR TIME SERIES DATABASE

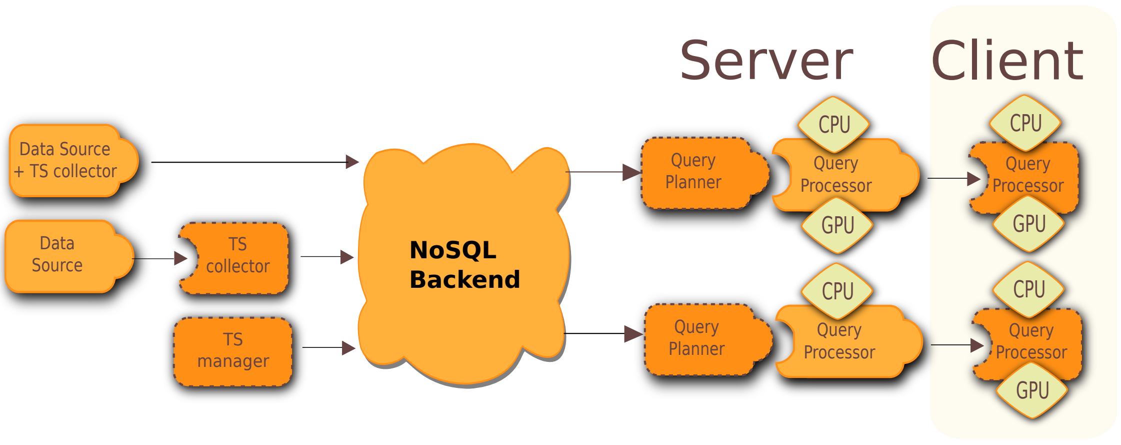


Figure 3.1: Time series database system overview

In this dissertation problems related to time series databases with GPGPU support are studied.

A typical time series database consists of three layers: a data storage, a data insertion module and a querying engine. The architecture described below (see Figure 3.1 for overview) was the starting point for the rest of the research.

To achieve high performance and scalability one of the *Big Table* compliant NoSQL databases may be utilized as a database storage. There are successful examples of using Hbase and Cassandra in this context [19, 55].

Data collection components are responsible for collecting data from different sources and inserting them into the database. This process must have a continuous character since *data sources* are generating new values permanently. Usually this is a result of ongoing measurements or monitoring processes (the operating system, databases and web servers or network devices: see OpenTSDB monitoring tools [55]). The system should not limit possible data which can be stored and should store data in a lossless form. In real life application it is safer to assume that all processed time series are unevenly spaced. Time series are stored as pairs of a *timestamp* and a *numerical value* $(t_i, v_i)$. Each time series should be explicitly identified by metadata.

We assume that distributed *data collectors* (possibly many instances, which may work directly on data sources) will push the data directly into data storage. To ensure optimal data storage a *TS manager* should be used to compact and compresses data. Furthermore, we assume that time series are divided into disjoint subsequences and stored in separate records.

Users may execute queries using any instance of *Query planner* which creates a heterogeneous query plan. The created query plan may utilize heterogeneous query processor GPUs and/or CPUs. The query planner should be aware of the load of subsequent units in the environment. The generated query plan should also take into account many factors: the CPU and GPU speed, the available memory, the bandwidth of memory access, the size of data, the compression ratio and the network connection speed. Due to all these factors not all requests may accelerate through processing on a GPU. The queries should be executed on the GPU only when the amount of time related to the use of the GPU is amortized by faster data processing [13].

As processing queries in a client-server architecture can bring tangible benefits [28], it is worth to consider a query planner capable of using the client computing resources.

## 3.2 LIGHTWEIGHT COMPRESSION FRAMEWORK FOR CPU/GPU

The results of studies on the use of lightweight compression in GPGPU computing are described in publications [58–60], which can be found in Appendix A.1 on page 27, Appendix A.3 on page 47 and Appendix A.2 on page 37, respectively.

The usage of lightweight compression methods in database systems may greatly improve their performance [82, 83]. Time series database systems are no except and lightweight compression methods may greatly boost performance of such systems. This is because they have serious impact on the size and the performance of the underlying time series database storage. Lightweight compression algorithms are designed in such a way that the decompression throughput is higher than the disk data throughput. The increase in the performance of a database system is associated with the increase of the data throughput.

Moreover, lightweight compression methods may improve the performance of GPGPU computing in database applications [25, 58–60]. That is mainly due to the reduced data transfer overhead from the storage to RAM and from RAM to the global device's memory space. Furthermore, in some applications it is possible to decompress the data on the fly (only for computation time). Thus, it is possible to reduce the overall memory access time on the GPU device. This approach may be also used in more general data intensive GPGPU computations.

Lightweight compression algorithms are mainly designed to improve the data transfer throughput and not the compressed data size. Therefore, it is worth to consider a cascaded lightweight compression planner which uses multiple lightweight compression algorithms in order to improve the compression ratio [25, 60]. As cascaded compression plans are more demanding computationally and may reduce the data decompression throughput, it is crucial to find a balance between the compression ratio and the decompression overhead.

Obviously, different time series will have different characteristics. Moreover, different subsequences of the same time series may have diverse characteristic. Therefore, the compression planner should be able to apply different compression plans for different time series subsequences [60].

In conclusion, in [60] we have significantly improved results presented by Fang et. al. in [25] by:

- introducing three patched lightweight compression algorithms to the GPU platform,
- enhancing some of earlier algorithms implementations from [25] (i. e. allowing any encoding bit lengths from interval $(2, 3, \ldots, 32)$ as opposed to results from [25], where only bit lengths of multiples of a byte were considered).

Furthermore, we have presented a concept of a dynamical lightweight compression planner for time series data [58] which utilizes time series characteristics. Afterwards, we have shown that lightweight compression may drastically improve the use of GPGPU to date intensive applications and, in particular, in time series processing [58–60]. Finally, we have shown that in some applications it is possible to improve the overall memory access time on a GPU device by using decompression on the fly.

## 3.3 TIME SERIES QUERIES PROCESSING FOR CPU/GPU

We consider time series query processing in the following publications [58, 59, 61, 62] (Appendix A.5 on page 69, Appendix A.1 on page 27, Appendix A.3 on page 47 and Appendix A.6 on page 81, respectively).

Measuring similarity between two time series is an important subproblem of a wider range of time series data mining problems, as almost every time series mining task requires a subtle notion of similarity between series. For instance, *query by content*, K*-nearest neighbours query*, $\epsilon$*-range query* are known tasks in time series data mining that depend on a similarity measure. In the paper [58] we discuss *whole series matching* on GPU device by using Hamming distance as a similarity function. We have found that despite the fact that implementation of Hamming distance function on a GPU is highly computationally efficient, the data transfer from RAM to GPU is a major problem and spoils the performance of the GPU in such applications. For this reason, it was necessary to use lightweight compression methods (see Section 3.2) to eliminate the data transfer bottleneck [58]. We also have been able to successfully experiment with other similarity measures which are known to have efficient implementations on a GPU, in particular Euclidean distance and Dynamic Time Warping [57, 68]. In both cases, we have been able to reduce the data transfer cost. However, DTW calculation is so computationally intensive that the reduced data transfer has a much smaller effect on the performance.

In [60] we discuss GPU efficient implementation of algorithms for time series preprocessing and transformation algorithms based on OpenTSDB query language [55]. In the above mentioned paper we present algorithms for downsampling, interpolation and aggregation, tasks commonly used in many time series databases. As in the previous case, in order to fully exploit the potential of the GPU computation, the data transfer issue should be addressed. Once again the usage of light compression turns out to be an effective solution to the data transfer bottleneck.

Despite that research presented in [61, 62] focuses primarily on biomonitoring systems, it is worth to notice that the underneath system operates on a time series database. Various time series data mining processing techniques are used in this system, e. g. noise reduction using wavelet filters, pattern matching, wavelets based features extraction and classification. Although currently our biomonitoring system is not supported on the GPU platform, there exist efficient GPU implementations for some of the missing components. In particular, discrete wavelet transform was presented in [79], k-NN search was discussed in [29, 30].

The results of this research will serve in the future as a starting point for development of an advanced database system for time series with GPU support.

## 3.4 A BI-OBJECTIVE OPTIMIZATION FRAMEWORK FOR HETEROGENEOUS CPU/GPU QUERY PLANS

Research on heterogeneous CPU/GPU query plans optimization was presented in [63] in Appendix A.4 on page 55.

Previous studies suggest that numerous database tasks may be accelerated by the usage of the GPU processing [10–15, 58–60, 78]. However, there are some issues that have to be solved. We suggested a solution to the obligatory data transfer from a CPU to a GPU issue by introducing lightweight compression framework in the previous section (see Section 3.2).

Yet, this does not solve all problems. As GPU is mostly optimized for numerical computation, only selected operations will benefit from GPU. Furthermore, small data sets are also a problem. It is often the case that the performance gain is negligible (or even negative).

Therefore, it is crucial to use joint processing capabilities of both CPU and GPU [63]. We investigate heterogeneous CPU/GPU computation and discuss optimisation methods that seek the balance between these two platforms. The method is based on heuristic search for bi-objective Pareto optimal execution plans in the presence of multiple concurrent queries. Our query planner uses one of three supported optimization types:

- time (minimize processing time),
- cost (minimize processing cost),
- bi-objective (combine time and cost objectives).

Cost objective is based on the commodity market model where devices are interpreted as producers and queries are interpreted as consumers. The price of the resources is controlled by supply-and-demand laws. The simulation experiment results are promising. We have achieved good load balancing combined with better optimization results than in the previous research [63]. Furthermore, our solution also offers lower time complexity and higher accuracy than other known methods.

## 3.5 EXEMPLARY APPLICATIONS

Application of time series database in biomonitoring systems was described in two publications. Paper [62] is an extended work of the research presented in [61] (see Appendix A.6 on page 81 and Appendix A.5 on page 69, respectively).

Water pollution monitoring is one of the most critical aspects of environmental and health care. Many existing monitoring systems work only for a narrow range of substances and do not provide continuous operation. Because of this, systems based on life organisms (Biological Early Warning Systems) increasingly gain interest and popularity.

Such systems use long-term observations of bioindicator activity for monitoring purposes. Some species of mussels are well-known bioindicators and *Dreissena polymorpha* is one of them. In these studies, we have collected observations of behaviour of *Dreissena polymorpha* and analysed them in order to detect any emerging threats. As observations of mussels behaviour are done by measuring the gap between the valves, we used a time series database system to store, process and mine the collected data. We propose an efficient framework for automatic classification of control and stressful conditions, distinguishing between stressful conditions and basic behaviour patterns extraction.

To accomplish this, we use a time series database and various data mining and processing techniques like wavelet filters, pattern matching, wavelets features extraction and classification.

## 3.6 conclusion

In this dissertation, query optimization in heterogeneous CPU/GPU environment for time series databases was discussed.

We have demonstrated that GPU used as a coprocessor in a time series database may significantly improve query preprocessing performance [58, 60]. However, performance boost is only possible if several issues related to GPU computing are addressed.

One of the problems is the obligatory time consuming memory transfer between CPU and GPU which precedes most GPU computations. In this dissertation, we consider usage of lightweight compression methods to minimize memory transfer and, therefore, to improve processing performance. We discuss lightweight compression algorithms, implementation details and their applications in the context of GPGPU [58–60]. We propose new GPU implementations and extensions of some known algorithms. Furthermore, we propose a data adaptive lightweight compression planner [60]. Resulting compression ratios and decompression bandwidth of proposed solution are an attractive option for GPU supported databases.

Another issue bound to GPU as a coprocessor in database environment is that not all processing task actually fit GPU architecture. This may be related to tasks' size or type. In this dissertation, we elaborate on computation and optimisation methods in heterogeneous CPU/GPU environments [63]. Our approach is based on an economic model which mimics the commodity market and we use bi-objective optimization, combining economic based cost objective with estimated execution time objective. We propose an heuristic search of bi-objective optimal execution plans. The preliminary results are very promising and offer good load balancing of the simulated devices combined with better optimization results than in other approaches [63].

Finally, we discuss an example application of time series database in an biological experiment. In this trial, we observe change over time of the gap between Zebra mussel valves. Those observations are naturally in the form of time series. We propose a new algorithm based on wavelets and kernel methods which extracts mussel behavioural events by analysing resulting time series [61, 62]. Automatic extraction of behaviour events greatly improves research on Zebra mussels behaviour. Moreover, we extend applications of this algorithm by proposing an efficient framework for automatic classification of mussel behaviour depending on surrounding environment state [62]. This is an important step towards the creation of an advanced environmental biomonitoring system based on above-mentioned mussel species.

## 3.7 FUTURE WORK

In the future, we plan to further expand the methods presented in this dissertation.

In the context of time series databases we plan to extend query processing methods on GPU to support advanced time series querying and data mining tasks. We also plan to adapt the proposed methods to support monitoring in SCADA systems.

Moreover, we plan an extensive study on lightweight compression algorithms applications in general context. In particular, we are interested in investigating usage of those algorithms on various SIMD architectures. We will also continue research on the proposed dynamic compression planner, in particular, we plan to add support for processing of partially decompressed data and support for different data structures (graphs for example).

Furthermore, we plan an extended evaluation of the heterogeneous CPU/GPU framework, along with examination of parameters' influence on the model behaviour and assessment against Hybrid Query challenges. Another interesting field is an extension of this model beyond CPU/GPU co-processing.

Finally, we plan to expand research on methods associated with Zebra mussels behaviour analysis with the goal to develop an advanced water biomonitoring system.

# A APPENDIX: CONTRIBUTION PUBLICATIONS

## Contents

# Improving Efficiency of Data Intensive Applications on GPU Using Lightweight Compression

Piotr Przymus[1] and Krzysztof Kaczmarski[2]

[1] Nicolaus Copernicus University, Chopina 12/18, Toruń, Poland
eror@umk.mat.pl
[2] Warsaw University of Technology, Plac Politechniki 1, 00-661 Warszawa, Poland
k.kaczmarski@mini.pw.edu.pl

**Abstract.** In many scientific and industrial applications GPGPU (General-Purpose Computing on Graphics Processing Units) programming reported excellent speed-up when compared to traditional CPU (central processing unit) based libraries. However, for data intensive applications this benefit may be much smaller or may completely disappear due to time consuming memory transfers. Up to now, gain from processing on the GPU was noticeable only for problems where data transfer could be compensated by calculations, which usually mean large data sets and complex computations. This paper evaluates a new method of data decompression directly in GPU shared memory which minimizes data transfers on the path from disk, through main memory, global GPU device memory, to GPU processor. The method is successfully applied to pattern matching problems. Results of experiments show considerable speed improvement for large and small data volumes which is a significant step forward in GPGPU computing.

**Keywords:** lightweight compression, data-intensive computations, GPU, CUDA.

## 1 Introduction

In all branches of science and industry amount of data which needs to be processed increase every year with enormous speed. Often this analysis involve uncomplicated algorithms but working on large data sets which in most cases cannot be efficiently reduced. This kind of applications are called data-intensive and are characterized by the following properties:

1. data itself, its size, complexity or rate of acquisition is the biggest problem;
2. require fast data access and minimization of data movement;
3. expects high, preferably linear, scalability of both hardware and software platform.

One of the most typical solutions to process large volumes of data is the map-reduce algorithm which gained huge popularity due to its simplicity, scalability and distributed nature. It is designed to perform large scale computations which may last from seconds to weeks or longer and involve from several to hundreds or thousands of machines processing together peta-bytes of data.

On the opposite side we can find problems which reside in a single machine but are large enough to require high processing power to get results within milliseconds. GPU programming offers tremendous processing power and excellent scalability with increasing number of parallel threads but with several other limitations. One of them is obligatory data transfer between RAM (random-access memory) and the computing GPU processor which generates additional cost of computations when compared to a pure CPU-based solution. This barrier can make all GPU computations unsatisfactory especially for smaller problems.

Time series matching is a popular example of this kind of data intensive applications in which a user may expect ultra fast results. Therefore in the rest of this work we use this example as a proof of concept application.

### 1.1 Motivation

Goal of this work is to improve efficiency of data transfer between disk, through RAM, global GPU memory and processing unit, which is often a bottleneck of many algorithms and gain noticeable higher speed up of algorithms when compared to classical CPU programming. We focus on memory bound data intensive applications since many computation intensive applications already proved to be much faster when properly implemented in parallel GPU algorithms.

Let us consider a problem of matching whole patterns in time series. As a distance function we can use the Hamming distance. Due to the lack of complex calculations main limitation of this problem is data transfer. To facilitate the transfer of data we can use either hardware solutions or try to reduce the data size by compressing or eliminating data. Since the hardware is expensive and the elimination of data is not always possible, data compression is usually the only option. Classic compression algorithms are computationally expensive (gain from the transfer data does not compensate the calculations [1]) and difficult to implement on the GPU [2]. Alternatives such as lightweight compression algorithms which are successfully used for the CPU are therefore very attractive.

### 1.2 Related Works

Lossless data compression is a common technique for reducing data transfers. In the context of SIMD (Single Instruction Multiple Data) computations data compression was successfully utilized by [3] in tree searching to increase efficiency of cache line transfer between memory and processor. The authors indicate that GPU implementation of the same search algorithm is computational bound and cannot be improved by compression. In [4] authors present interesting study of different compression techniques for WWW data in order to achieve querying speed up. A general solution for data intensive applications by cache compression is discussed in [1]. Obviously efficiency may be increased only if decompression speed is higher than I/O operation. In our case we show that decoding is really much faster and eliminates this memory bottleneck. The compression schemes proposed by Zukowski et al. [1] offer good trade-off between compression time and encoded data size and what is more important are designed especially for super scalar processors which means also very good properties for GPU. Delbru et al. [5] develop a new Adaptive Frame of Reference compression algorithm in order to achieve significant speed up of compression time, which in turn allows for effective updates of data.

All these works are in fact entry points to our solution. Our contribution is that the compression methods have been redesigned for GPU architecture and may offer excellent encoding and decoding speed for all compression ratios. Our solution may be utilised in any data intensive algorithms involving integer or fixed-decimal number data. We also explain how this method may be extended to other algorithms.

As the preliminary work we checked if lightweight compression algorithms can be used for different kinds of common time series databases. We collected data from various data sets covering: star light curve (TR1, TR2 – [6]), CFD (contract for difference) and stock quotations for AUDJPY, EURUSD, Shenzen Development Bank respectively (IN1, IN2, TS1 – [7,8]), radioactivity in the ground at 2 hourly intervals over one year (TS2 – [8]) and sample ECG (electrocardiogram) data (PH1,PH2 – [9]). The data used in this experiment is the data with fixed decimal precision that can be stored as integers, which allows to use lightweight compression. Efficiency of conventional compression algorithms (gzip, bzip2 - with default settings) compared to the lightweight compression methods (PFOR, PFOR-DIFF) is presented in figure 1. We can notice that lightweight compression, although not achieving as high compression ratio as gzip, may also obtain similar results, bzip2 is always much better but also significantly slower. This is also proved by L. Wu et al. [2] who showed that conventional compression is too slow to increase overall algorithm efficiency for GPU.

Due to its specificity, time series usually have a good compression ratio (Fig. 1). The most important benefits of compression can be summarized as follows:

- shorter time to load data from disk
- shorter time to copy data from RAM to device
- possibility to fit larger set of data directly on the GPU (GPU DRAM is often limited)
- reduced data size in storage

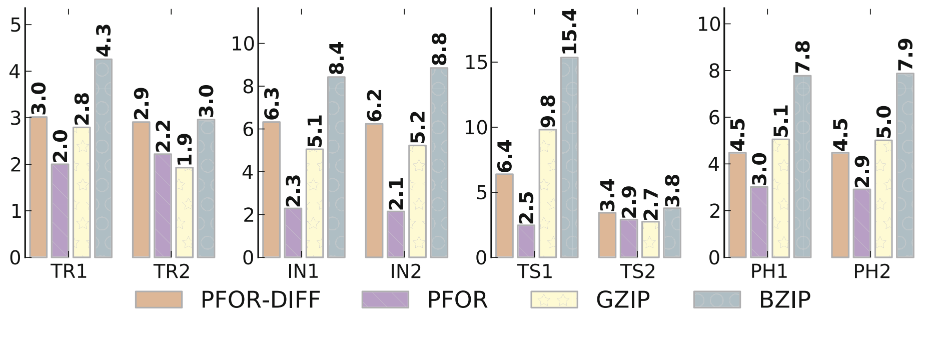


**Fig. 1.** Achieved compression level of lightweight compression and conventional compression methods for various datasets (higher is better). Data sets: TR1, TR2 – star light curve; IN1, IN2, TS1 – CFD and stock quotations; TS2 – radioactivity in the ground; PH1, PH2 – ECG data.

We conclude that lightweight compression may be used in time series data intensive applications achieving compression ratio from 2 to 7. This initial investigation lets us to predict that data transfer cost accompanied with decompression time may be noticeably decreased.

### 1.3 Research Hypothesis and Methodology

The main research hypothesis for this project is as follows.

- Data-intensive applications may increase their efficiency by utilization of lightweight compression methods in GPU global and shared memory.
- Cost of data decompression can be amortised by fewer global memory reads.
- Additional benefit may be obtained by proper utilisation of shared memory decompression.
- Data-intensive applications may benefit from GPU computing when applied for smaller data sets than without the method.

In order to verify the hypotheses we implement a prototype which will be used as a proof of concept equipped with several real-time and real-data measurements performed by fine grained timers and memory access analysis done by a professional profiler. Checking the hypotheses will involve a few kinds of experiments including data intensive application using:

1. GPU algorithm without decompression compared to CPU algorithm without decompression. This will allow to estimate possible general speed up of an algorithm when run on a GPU and also will be a starting point for other experiments by registering time necessary to perform pure computations without any decompression overhead.
2. GPU algorithm with decompression in global memory compared to CPU algorithm with decompression. This will show potential speed up when data transfer is minimised by data compression.
3. GPU algorithms with decompression in shared memory compared to GPU with decompression in global memory and CPU algorithm with decompression. This will show the speed up when minimising GPU global memory reads.

As an initial set up for this research we use time series matching problems as an exemplar of data intensive applications. Because of efficiency requirements mentioned in the previous section we utilise lightweight compression methods: PFOR and FOR-DIFF algorithms.

## 2 Method Description

### 2.1 Lightweight Compression

In this section we discuss the algorithm FOR and FOR-DIFF in different variations, as well as discuss problems and solutions when using this approach on the GPU. FOR determines range of values for the frame, and then maps the values in this interval using a minimum number of bits needed to distinguish the data [10]. A common practice is to convert the data in the frame to the interval $\{0,\ldots,max-min\}$. In this situation, we need exactly $\lceil log_2(max-min+1)\rceil$ bits to encode each value in the frame.

The main advantage of the FOR algorithm is the fact that compression and decompression are highly effective on GPU because these routines contain no branching-conditions, which decrease parallelism of SIMD operations. Additionally functions are loop-unrolled and use only shift and mask operations. This implies that there are dedicated compression and decompression routines prepared for every bit encoding length.

The compression algorithm works as follows, for the data frame of length $n$ loop is performed each $m$ values (where $m$ is a multiple of 8) and compressed using the same function at each iteration step. Decompression is similar to compression. We iterate through the $m$-coded values, and we use a function that decodes $m$ values.

FOR-DIFF algorithm is similar to FOR, however, stores the differences between successive data points in frame. Compression needs to calculate the difference, then compresses them using the FOR compression scheme. Decompression begins by decompressing and then reconstructs the original data from the differences. This approach can significantly improve the compression ratio for certain types of data.

The main drawback of FOR is that it is prone to outliers in the data frame. For example, for the frame $\{1,2,3,3,2,2,2,3,3,1,1,64,2,3,1,1\}$, if not the value 64 we could use the $\lceil log_2(3-1+1)\rceil = 2$ bits to encode the frame, but due to the outlier we have to use 6-bit encoding ($\lceil log_2(64-1+1)\rceil$) thus wasting 4 bits for each element.

Solution to the problem of outliers has been proposed in the work [1] in the modified version of the algorithm, called Patched FOR or PFOR. In this version of the algorithm outliers are stored as exceptions. PFOR first selects a new range mapping for the data in order to minimise the size of compressed data frames taking into account the space needed to store exceptions. Therefore, compressed block consists of two sections, within the first section the compressed data are kept and in the second exceptions are stored (encoded using 8, 16 or 32 bits). Unused slots for exceptions in the first section are used to hold the offset of the following exception in the data in order to create linked list, when there is no space to store the offset of the next exception, a *compulsive exception* is created [1]. For large blocks of data, the linked lists approach may fail because the exceptions may appear sparse thus generate a large number of compulsory exceptions. To minimise the problem of various solutions have been proposed, such as reducing the frame size [1], or algorithms that do not generate compulsive exceptions, such as Adaptive FOR [5] or modified version of PFOR [4]. Our publication is based on PFOR algorithm presented in [4]. Compressed block consists of three parts: the first in which compressed data are kept, second section where exceptions offsets are stored, and the last section which holds remainders of exceptions. When the outlier is processed, its position is preserved in an offset array, and the value divided into bits that are stored in the first section and the remainder to be retained in the third. Second and third section are then compressed using FOR(separately).

Decompression proceeds as follows: first decompresses the data section. Then decompresses the offset and exceptions array. Then iterates over the array offset, and restore the value of by applying patch from exception array.

## 2.2 GPU Implementation

To be able to decompress a data frame, decompression functions must be specified (for block of values, positions and supplements). In the case of the CPU one can use function pointers. On GPU this is dependent on the GPU computation capabilities (cc). For cc lower than the 2.x is not possible to use function pointers, which were introduced in version 2.x and higher. So far we used solution compatible with both architectures. In future work a version that uses features introduced in 2.x cards may simplify the code and make it more flexible.

Our current implementation is based on macros and requires information about the compressed data at compile time. This is not a serious limitation in the case of data that we analysed, as the optimum compression parameters were found to be constant within particular sets of data. Usual practice is to determine the optimal compression parameters based on sample data [1] and use them for the rest of the data set.

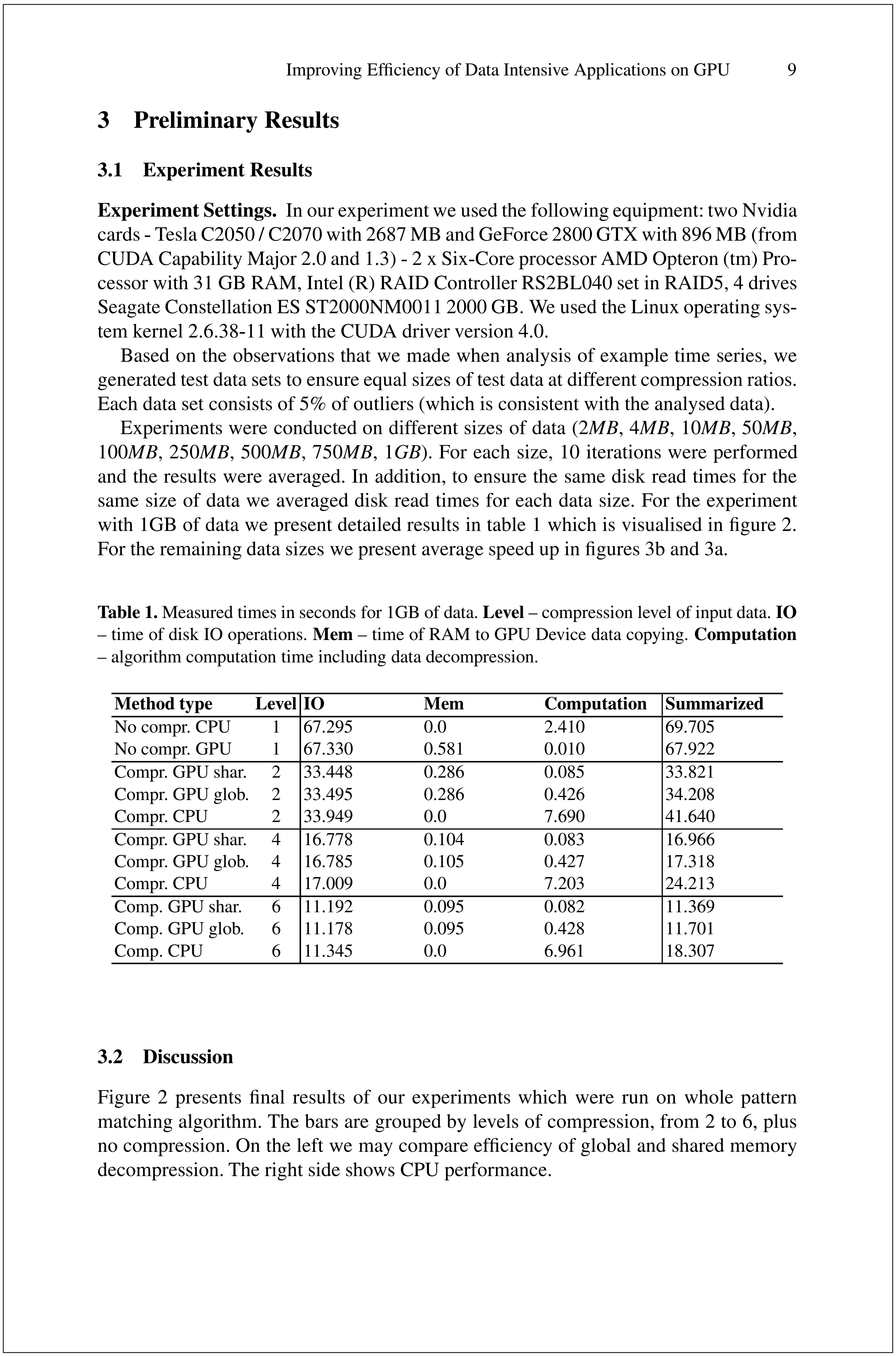
Another challenge was the issue of optimal reads of global memory during decompression: first decompression threads do not form coalesced reads leading to a drop in performance, and secondly the CUDA architecture can not cope well with the types of readings less than 32 bits in length. Our solution involves packaging data in texture memory, which solves the first problem. The second problem is solved by casting compressed data (char array) to array of integers. Decompression requires the reverse conversion. Texture, however, introduces some limitations, the maximum size is $2^{27}$ elements in case of one dimensional array. In future we plan to abolish the need for packaging data in the texture by modifying the compression and decompression algorithm. In this way readings of compressed data will allow for coalesced reads.

We prepared two versions of the experiments for GPU. In the first one we decompresses into the shared memory and in the second one into global memory. Algorithm based on global memory, as expected, is considerably slower than the algorithm based on shared memory (global memory is hundreds of times slower than shared), but this is still good choice for the considered experiment. On the other hand using shared memory is not always possible, so the algorithm based on global memory is still an alternative.

In the case of PFOR-DIFF algorithm we still need to perform reconstruction step, which involves iterating over decompressed buffer. In our opinion the best solution for this step, is to use properly implemented parallel prefix sum.

To decompress $m$ (where $m$ is a multiple of 8) values, we use $m/8$ threads. Each of the threads decompresses 8 values. Assuming that we have $k$ exceptions (where $k$ is a multiple of 8) we use the $k/8$ threads to decompress the offset and exception array. Then we patch 8 values as indicated in offset array using exceptions. This scheme is easily customisable when we have fewer threads. For performance reasons, we still need an array of 24 integers (safely located within threads registers) - ensuring best performance. After the decompression phase, array can be safely used for other purposes. For the convenience we have a macro that entered at the beginning of the kernel will prepare the appropriate code for uncompressing data, making use of decompression in other solutions easy.

We prepared uncompressing data code for both algorithms, ie. PFOR and PFOR-DIFF. In the experimental part we used the PFOR-DIFF algorithm as it is computationally more complex.



## 3 Preliminary Results

### 3.1 Experiment Results

**Experiment Settings.** In our experiment we used the following equipment: two Nvidia cards - Tesla C2050 / C2070 with 2687 MB and GeForce 2800 GTX with 896 MB (from CUDA Capability Major 2.0 and 1.3) - 2 x Six-Core processor AMD Opteron (tm) Processor with 31 GB RAM, Intel (R) RAID Controller RS2BL040 set in RAID5, 4 drives Seagate Constellation ES ST2000NM0011 2000 GB. We used the Linux operating system kernel 2.6.38-11 with the CUDA driver version 4.0.

Based on the observations that we made when analysis of example time series, we generated test data sets to ensure equal sizes of test data at different compression ratios. Each data set consists of 5% of outliers (which is consistent with the analysed data).

Experiments were conducted on different sizes of data (2$MB$, 4$MB$, 10$MB$, 50$MB$, 100$MB$, 250$MB$, 500$MB$, 750$MB$, 1$GB$). For each size, 10 iterations were performed and the results were averaged. In addition, to ensure the same disk read times for the same size of data we averaged disk read times for each data size. For the experiment with 1GB of data we present detailed results in table 1 which is visualised in figure 2. For the remaining data sizes we present average speed up in figures 3b and 3a.

**Table 1.** Measured times in seconds for 1GB of data. **Level** – compression level of input data. **IO** – time of disk IO operations. **Mem** – time of RAM to GPU Device data copying. **Computation** – algorithm computation time including data decompression.

| Method type | Level | IO | Mem | Computation | Summarized |
|---|---|---|---|---|---|
| No compr. CPU | 1 | 67.295 | 0.0 | 2.410 | 69.705 |
| No compr. GPU | 1 | 67.330 | 0.581 | 0.010 | 67.922 |
| Compr. GPU shar. | 2 | 33.448 | 0.286 | 0.085 | 33.821 |
| Compr. GPU glob. | 2 | 33.495 | 0.286 | 0.426 | 34.208 |
| Compr. CPU | 2 | 33.949 | 0.0 | 7.690 | 41.640 |
| Compr. GPU shar. | 4 | 16.778 | 0.104 | 0.083 | 16.966 |
| Compr. GPU glob. | 4 | 16.785 | 0.105 | 0.427 | 17.318 |
| Compr. CPU | 4 | 17.009 | 0.0 | 7.203 | 24.213 |
| Comp. GPU shar. | 6 | 11.192 | 0.095 | 0.082 | 11.369 |
| Comp. GPU glob. | 6 | 11.178 | 0.095 | 0.428 | 11.701 |
| Comp. CPU | 6 | 11.345 | 0.0 | 6.961 | 18.307 |

### 3.2 Discussion

Figure 2 presents final results of our experiments which were run on whole pattern matching algorithm. The bars are grouped by levels of compression, from 2 to 6, plus no compression. On the left we may compare efficiency of global and shared memory decompression. The right side shows CPU performance.

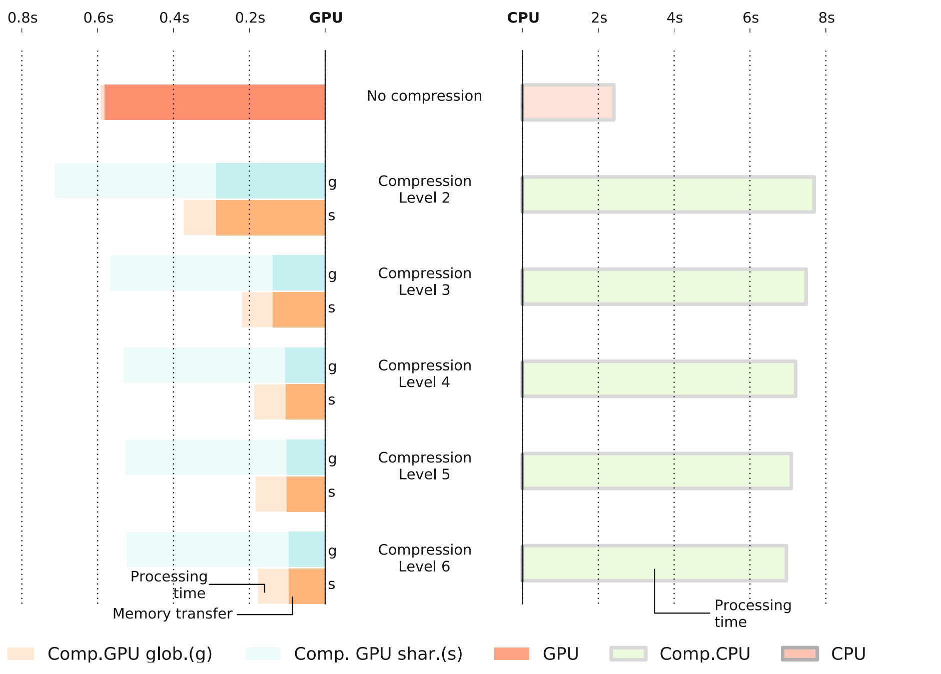


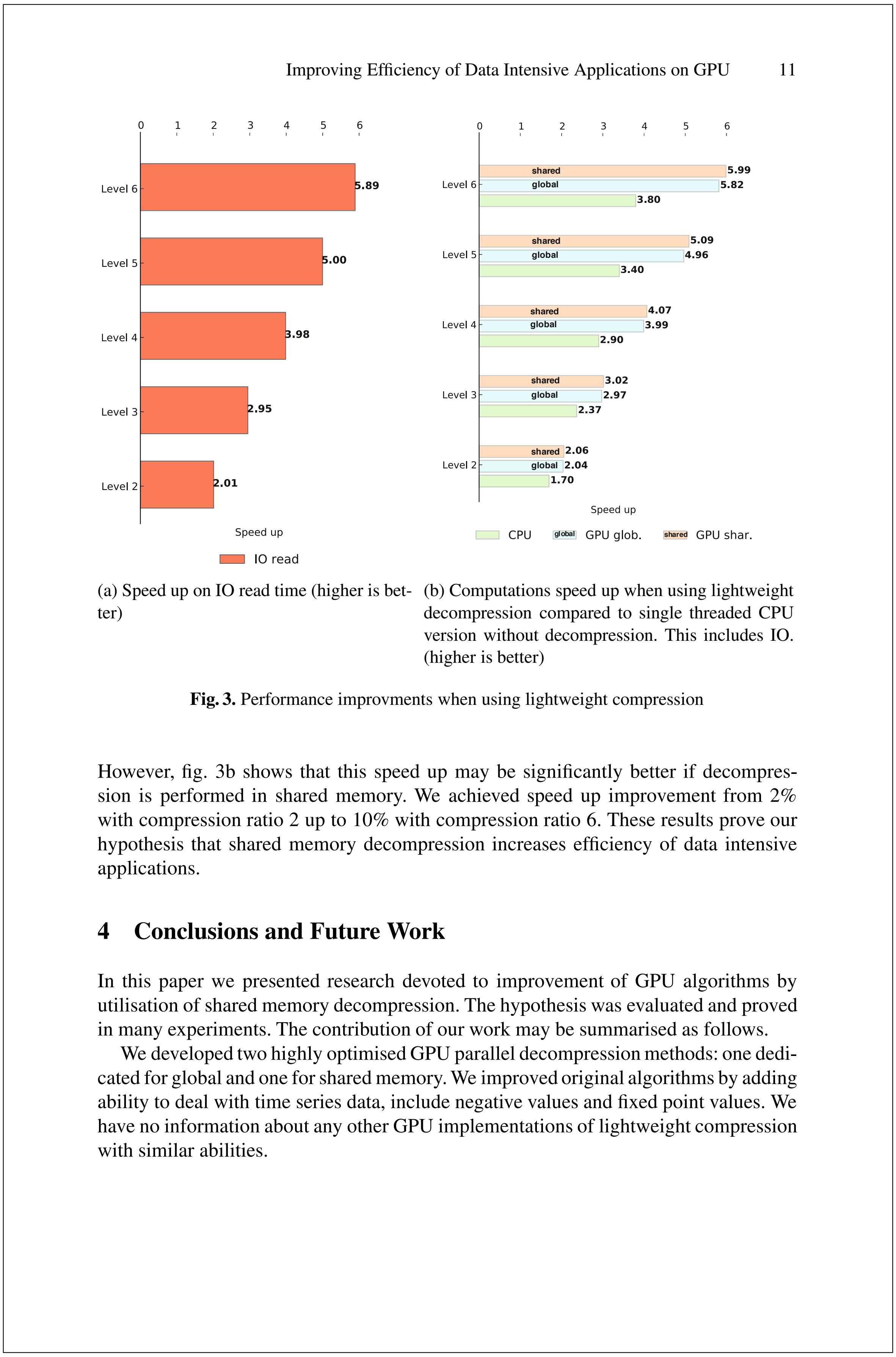
**Fig. 2.** Performance of a whole pattern matching algorithm with and without compression for 1GB of data, excluding IO times (lower is better). g – global memory decompression, s – shared memory decompression.

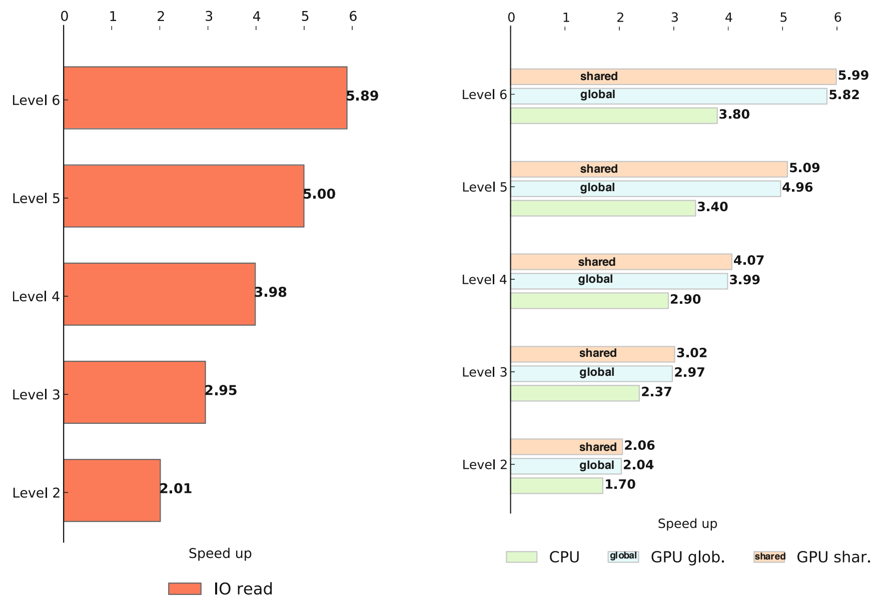
CPU performance is highly influenced by data decompression time. We can see that decompression takes from 4 to almost 6 seconds which is about 200% to 300% of the algorithm without decompression. The only possible improvement can be then observed on IO operations.

Thanks to ultra fast decompression GPU implementation of the algorithm performs much better. We can observe that for compression level 2 decompression in shared memory improved overall execution time by almost 2 times. For level 6 it is almost 3 times better. Decompression in global memory although also very fast can be slower for lower levels.

We must point out here that architecture of GPU shared memory limits possible number of algorithms which may use shared memory decompression method. Currently, subsequent kernel calls is the only method for global synchronisation. However, GPU cannot store shared memory state between kernel calls. As the consequence data decompression and algorithm execution must be done in single kernel. We are conscious that this may be improved by more complex kernels and data decompression strategy which will be addressed in our future research.

Figures 3a and 3b show interesting results concerning our shared memory decompression method. First we must notice that data compression alone increases application speed by reducing necessary data transfer. Fig. 3a demonstrates that pure IO operation speed up corresponds data compression ratio, which is an obvious result.





(a) Speed up on IO read time (higher is better)

(b) Computations speed up when using lightweight decompression compared to single threaded CPU version without decompression. This includes IO. (higher is better)

**Fig. 3.** Performance improvments when using lightweight compression

However, fig. 3b shows that this speed up may be significantly better if decompression is performed in shared memory. We achieved speed up improvement from 2% with compression ratio 2 up to 10% with compression ratio 6. These results prove our hypothesis that shared memory decompression increases efficiency of data intensive applications.

## 4 Conclusions and Future Work

In this paper we presented research devoted to improvement of GPU algorithms by utilisation of shared memory decompression. The hypothesis was evaluated and proved in many experiments. The contribution of our work may be summarised as follows.

We developed two highly optimised GPU parallel decompression methods: one dedicated for global and one for shared memory. We improved original algorithms by adding ability to deal with time series data, include negative values and fixed point values. We have no information about any other GPU implementations of lightweight compression with similar abilities.

We tested lightweight compression for time series and applied our novel methods to data intensive pattern matching mechanism. Our evaluation proved that the method significantly improved results when compared to other methods.

We showed that data decompression in GPU shared memory may generally improve performance of other data intensive applications due to ultra fast parallel decompression procedure since time saved by smaller data transfer is not wasted for decompression.

We analysed relationship between global and shared memory decompression showing that although shared memory may limit number of possible applications due to lack of global synchronisation mechanism, it significantly improves performance.

We plan to design a general purpose library which could be used in a similar way to CUDA Thrust library. Utilisation of common iterator pattern with memory blocks overlapping may offer interesting abilities breaking barrier of inter block threads communication. Such an iterator would require to define size of a shared memory buffer which would be common for more than one block in parallel threads execution. Description of such an extended iterator pattern for overlapping shared memory will be the next step of this research. During the preparation of this work, we managed to locate several problems of the current solution. In the future we also plan to prepare a new version of the algorithm which will allow for better use of CUDA 2.x computation capabilities and remove the constraints which we mentioned in the implementation description.

This work is a part of a larger project which aims to create a GPU based database for scientific data (such as time series, array, etc.).

## References

1. Zukowski, M., Heman, S., Nes, N., Boncz, P.: Super-scalar ram-cpu cache compression. In: Proc. of the 22nd Intern. Conf. on Data Engineering, ICDE 2006, pp. 59–59. IEEE (2006)
2. Wu, L., Storus, M., Cross, D.: Cs315a: Final project cuda wuda shuda: Cuda compression project. Technical report. Stanford University (March 2009)
3. Kim, C., Chhugani, J., Satish, N., Sedlar, E., Nguyen, A.D., Kaldewey, T., Lee, V.W., Brandt, S.A., Dubey, P.: Fast: fast architecture sensitive tree search on modern cpus and gpus. In: Proc. of the 2010 Intern. Conf. on Management of Data, pp. 339–350. ACM (2010)
4. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proc. of the 18th Intern. Conf. on World Wide Web, pp. 401–410. ACM (2009)
5. Delbru, R., Campinas, S., Samp, K., Tummarello, G.: Adaptive frame of reference for compressing inverted lists. Technical report. DERI – Digital Enterprise Research Institute (December 2010)
6. Harvard IIC. Data and search interface, time sries center (2012), `http://timemachine.iic.harvard.edu/`
7. Integral. Truefx (2012), `http://www.truefx.com/`
8. Hyndman, R.J.: Time series data library (2012), `http://robjhyndman.com/tsdl`
9. Goldberger, A.L., et al.: Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. Circulation 101(23), e215-e220
10. Goldstein, J., Ramakrishnan, R., Shaft, U.: Compressing relations and indexes. In: Proc. of the 14th Intern. Conf. on Data Engineering, pp. 370–379. IEEE (1998)

# Dynamic Compression Strategy for Time Series Database Using GPU[⋆]

Piotr Przymus[1] and Krzysztof Kaczmarski[2]

[1] Nicolaus Copernicus University, Poland
eror@umk.mat.pl
[2] Warsaw University of Technology, Poland
k.kaczmarski@mini.pw.edu.pl

**Abstract.** Nowadays, we can observe increasing interest in processing and exploration of time series. Growing volumes of data and needs of efficient processing pushed research in new directions. GPU devices combined with fast compression and decompression algorithms open new horizons for data intensive systems. In this paper we present improved cascaded compression mechanism for time series databases build on Big Table–like solution. We achieved extremely fast compression methods with good compression ratio.

**Keywords:** time series database, lightweight lossless compression, GPU, CUDA.

## 1 Introduction

Specialized time series databases play important role in industry storing monitoring data for analytical purposes. These systems are expected to process and store millions of data points per minute, 24 hours a day, seven days a week, generating terabytes of logs. Due to regression errors checking and early malfunction prediction these data must be kept with proper resolution including all details. Solutions like OpenTSDB [11], TempoDB [3] and others deal very well with these kind of tasks. Most of them work on a clone of Big Table approach from Google [5], a distributed hash table with mutual ability to write and read data in the same time.

Usually systems compress data before writing to a long-term storage. It is much more efficient to store data for some time in a memory or disk buffer and compress it before flushing to disk. This process is known as a table row rolling. Current systems like HBase [1], Casandra [6] and others offer compression optimization for entire column family. This kind of general purpose compression is not optimized for particular data being stored (i.e. various time series with different compression potential stored in one column family).

Similar problems appear in in-memory database systems. Solutions based on GPU processing (like ParStream [2]) tend to pack as many data into GPU devices global memory as possible. Efficient data compression method would significantly improve abilities of these systems. An average internet service with about 10 thousands of simultaneously working users may generate around 80GB of logs every day. After

[⋆] The project is funded by National Science Centre, decision DEC-2012/07/D/ST6/02483.

compression they could fit into two Nvidia Tesla devices where average query can be processed within seconds compared to minutes in case of standard systems.

In case of time series compression ratio could be improved by a method tuned to types of data including its variability, span, differences, etc. However, tuning time slows down compression and often cannot fit into time window available in real time monitoring systems. This paper describes a dynamic compression strategy planner for time series databases using GPU processors with reasonable processing time and compression ratios. What is even more important, the resulting compressed data block can be decompressed very quickly directly into the GPU memory additionally allowing for ultra fast query processing, what we discussed in our previous publication [12].

The main contribution of this work is:

- three new implementations of patched compression algorithms on GPU
- a new dynamic compression planner for lightweight compression methods
- categorization for compression methods and reduction of configuration space for optimal plan searching
- evaluation of the achieved results on real-life data

Section 2.1 presents a general view of the system, section 2.2 contains the main contribution of our work: the dynamically optimized compression system. Experimental runtime results are contained in section 3 while section 4 concludes.

## 1.1 Motivation and Related Work

Optimal data compression of time series is an interesting and widely analysed computational problem. Lossless methods often use some general purpose compression algorithms with several modifications according to knowledge gathered from data. On the other hand, lossy compression approximate data using, for instance, splines, piecewise linear approximation or extrema extraction [9]. For industrial monitoring, lossy compression cannot be used due to possible degradation of anomalies.

In case of lossless compression one can use common algorithms (ZIP, LZO) which tend to consume lot of computation resources [4,15] or lightweight methods which are faster but not so effective. Our dynamic method attempts to combine properties of both approaches: is lossless but much faster than common algorithms, offers good compression ratios and may be computed incrementally. Also ability to decompress values directly into the processor shared memory should improve GPU memory bandwidth and enable it to be used in many data intensive applications.

An important challenge is to improve compression factor with an acceptable processing time in case of variable sampling periods. Interesting results in the filed of lossless compression done on GPU were presented by Fang et al. [8]. Using a query planner it was possible to achieve significant improvement in overall query processing on GPU by reducing data transfer time from RAM to global device's memory space. The strategy applied in our work is based on statistics calculated from inserted data and used to find an optimal cascaded compression plan for the selected lightweight methods.

In a time series database we often observe data grouped into portions of very different characteristics. Optimal compression should be able to apply different compression plans for different time series and different time periods. Comparing to [8] and [4] we can achieve better results by using dynamic compression planning methods with automated compression tuning upon processed time series data.

## 2 Dynamically Optimized Compression System

### 2.1 Time Series Database Architecture

**General View** A typical time series database consists of three layers: data insertion module, data storage and querying engine. Our compression mechanism touches all the layers working as a middle tier between the data storage and the rest of the system. In this work we shall focus on data compression mechanism assuming that decompression used by the query engine is an obvious opposite process.

### 2.2 Data Insertion

**Data Collection.** The data acquisition from ongoing measurements, industrial processes monitoring [10], scientific experiments [13], stock quotes or any other financial and business intelligence sources has got continuous characteristic. These discrete observations $T$ are represented by pairs of a *timestamp* and a *numerical value* $(t_i, v_i)$ with the following assumptions: *a*) number of data points (timestamps and their values) in one time series should not be limited; *b*) each time series should be identified by a name which is often called a *metric name*; *c*) each time series can be additionally marked with a set of *tags* describing measurement details which together with metric name uniquely identifies time series; *d*) observations may not be done in constant time intervals or some points may be missing, which is probable in case of many real life data.

**Initial Buffering.** Due to optimization purposes, data sent to the data storage should be ordered and buffered into portions, minimizing necessary disk operations but also minimizing the distributed storage nodes intercommunication. Buffering also prepares data to be compressed and stored optimally in an archive. Simplicity of data model imposed separated column families for compressed and raw data. Time series are separately compacted into larger records (by a metric name and tags) containing a specified period of time (e.g. 15 minutes, 2 hours, 24 hours – depending on the number of observations). This step directly predeceases dynamic compression which is described in the next section.

### 2.3 Compression Algorithms

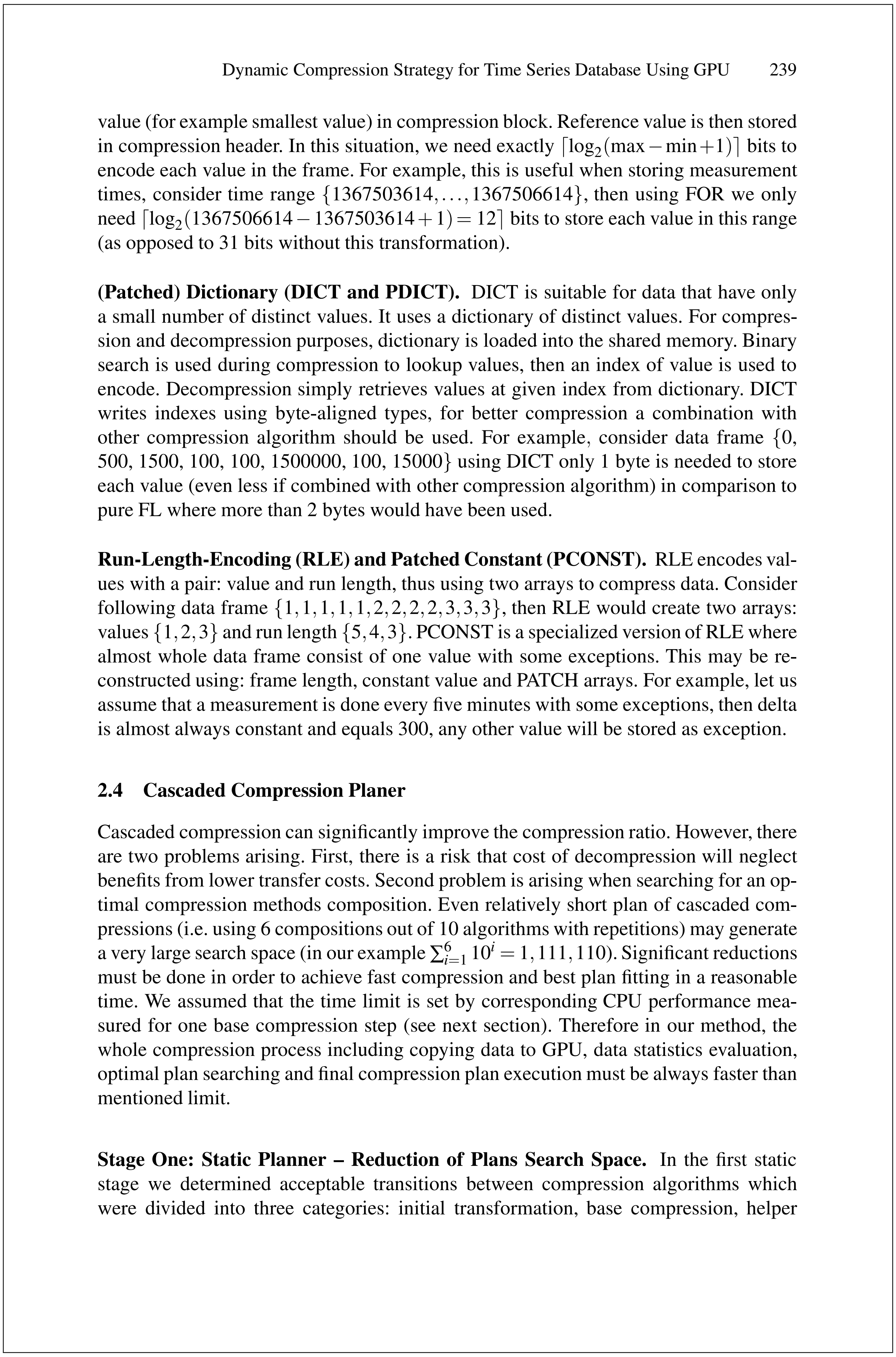
**Patched Lightweight Compression.** The main drawback of many lightweight compression schemes is that they are prone to outliers in the data frame. For example, consider following data frame $\{1,2,3,2,2,3,1,1,64,2,3,1,1\}$, one could use the 2 bits fixed-length compression to encode the frame, but due to the outlier (value 64) we have to use 6-bit fixed-length compression or more computationally intensive 4-bit dictionary compression. Solution to the problem of outliers has been proposed in [15] as a modification to three lightweight compression algorithms. The main idea was to store outliers as exceptions. Compressed block consists of two sections: the first keeps the compressed data and the second exceptions. Unused space for exceptions in the first section is used to hold the offset of the following exceptions in the data in order to create linked list, when there is no space to store the offset of the next exception, a *compulsive exception* is created [15]. For large blocks of data, the linked lists approach may fail because the exceptions may appear sparse thus generate a large number of compulsory exceptions. To minimise the problem various solutions have been proposed, such as reducing the frame size [15] or algorithms that do not generate compulsive exceptions [7,14]. The algorithms in this paper are based largely on those described by Yan [14]. In this version of the compression block is extended by two additional arrays - exceptions position and values. Decompression involves extracting data using the underlying decompression algorithm and then applying a patch (from exceptions values array) in the places specified by the exceptions positions. As exceptions are separated, data patching can be done in parallel. During compression, each thread manages two arrays for storing exception values and positions. After compression, each thread stores exceptions in the shared memory, similarly exceptions from shared memory are copied to the global memory. Patched version of algorithms are only selected if compression ratio improves. Otherwise non patched algorithms are used. Therefore complex exceptions treatment may be omitted speeding up the final compression.

**SCALE.** Converts float values to integer values by scaling. This solution can be used in case where values are stored with given precision. For example, CPU temperature 56.99 can be written as 5699. The scaling factor is stored in compression header.

**DELTA.** Stores the differences between successive data points in frame while the first value is stored in the compression header. Works well in case of sorted data, such as measurement times. For example, let us assume that every 5 minutes the CPU temperature is measured starting from 1367503614 to 1367506614 (Unix epoch timestamp notation), then this time range may be written as $\{300,\dots,300\}$.

**(Patched) Fixed-length Minimum Bit Encoding (PFL and FL).** FL and PFL compression works by encoding each element in the input with the same number of bits thus deleting leading zeros at the most significant bits in the bit representation. The number of bits required for the encoding is stored in the compression header. The main advantage of the FL algorithm (and its variants) is the fact that compression and decompression are highly effective on GPU because these routines contain no branching-conditions, which decrease parallelism of SIMD operations. For best efficiency dedicated compression and decompression routines are prepared for every bit encoding length with unrolled loops and using only shift and mask operations. Our implementation does not limit minimum encoding length to size of byte (as in [8]). Instead each thread (de)compresses block of eight values, thus allowing encoding with smaller number of bits. For example, consider following data frame $\{1,2,3,2,2,3,1,2,3,1,1\}$, one could use the 2 bits fixed-length compression to encode the frame.

**(Patched) Frame-Of-Reference (PFOR and FOR).** Works similarly to FL and PFL, except before compression it transforms each value into an offset from the reference



value (for example smallest value) in compression block. Reference value is then stored in compression header. In this situation, we need exactly $\lceil \log_2(\max - \min + 1) \rceil$ bits to encode each value in the frame. For example, this is useful when storing measurement times, consider time range $\{1367503614, \ldots, 1367506614\}$, then using FOR we only need $\lceil \log_2(1367506614 - 1367503614 + 1) = 12 \rceil$ bits to store each value in this range (as opposed to 31 bits without this transformation).

**(Patched) Dictionary (DICT and PDICT).** DICT is suitable for data that have only a small number of distinct values. It uses a dictionary of distinct values. For compression and decompression purposes, dictionary is loaded into the shared memory. Binary search is used during compression to lookup values, then an index of value is used to encode. Decompression simply retrieves values at given index from dictionary. DICT writes indexes using byte-aligned types, for better compression a combination with other compression algorithm should be used. For example, consider data frame {0, 500, 1500, 100, 100, 1500000, 100, 15000} using DICT only 1 byte is needed to store each value (even less if combined with other compression algorithm) in comparison to pure FL where more than 2 bytes would have been used.

**Run-Length-Encoding (RLE) and Patched Constant (PCONST).** RLE encodes values with a pair: value and run length, thus using two arrays to compress data. Consider following data frame $\{1,1,1,1,1,2,2,2,2,3,3,3\}$, then RLE would create two arrays: values $\{1,2,3\}$ and run length $\{5,4,3\}$. PCONST is a specialized version of RLE where almost whole data frame consist of one value with some exceptions. This may be reconstructed using: frame length, constant value and PATCH arrays. For example, let us assume that a measurement is done every five minutes with some exceptions, then delta is almost always constant and equals 300, any other value will be stored as exception.

### 2.4 Cascaded Compression Planer

Cascaded compression can significantly improve the compression ratio. However, there are two problems arising. First, there is a risk that cost of decompression will neglect benefits from lower transfer costs. Second problem is arising when searching for an optimal compression methods composition. Even relatively short plan of cascaded compressions (i.e. using 6 compositions out of 10 algorithms with repetitions) may generate a very large search space (in our example $\sum_{i=1}^{6} 10^i = 1,111,110$). Significant reductions must be done in order to achieve fast compression and best plan fitting in a reasonable time. We assumed that the time limit is set by corresponding CPU performance measured for one base compression step (see next section). Therefore in our method, the whole compression process including copying data to GPU, data statistics evaluation, optimal plan searching and final compression plan execution must be always faster than mentioned limit.

**Stage One: Static Planner – Reduction of Plans Search Space.** In the first static stage we determined acceptable transitions between compression algorithms which were divided into three categories: initial transformation, base compression, helper compression. The complete compression schema is always composed of algorithms selected from these ordered categories with the following purposes:

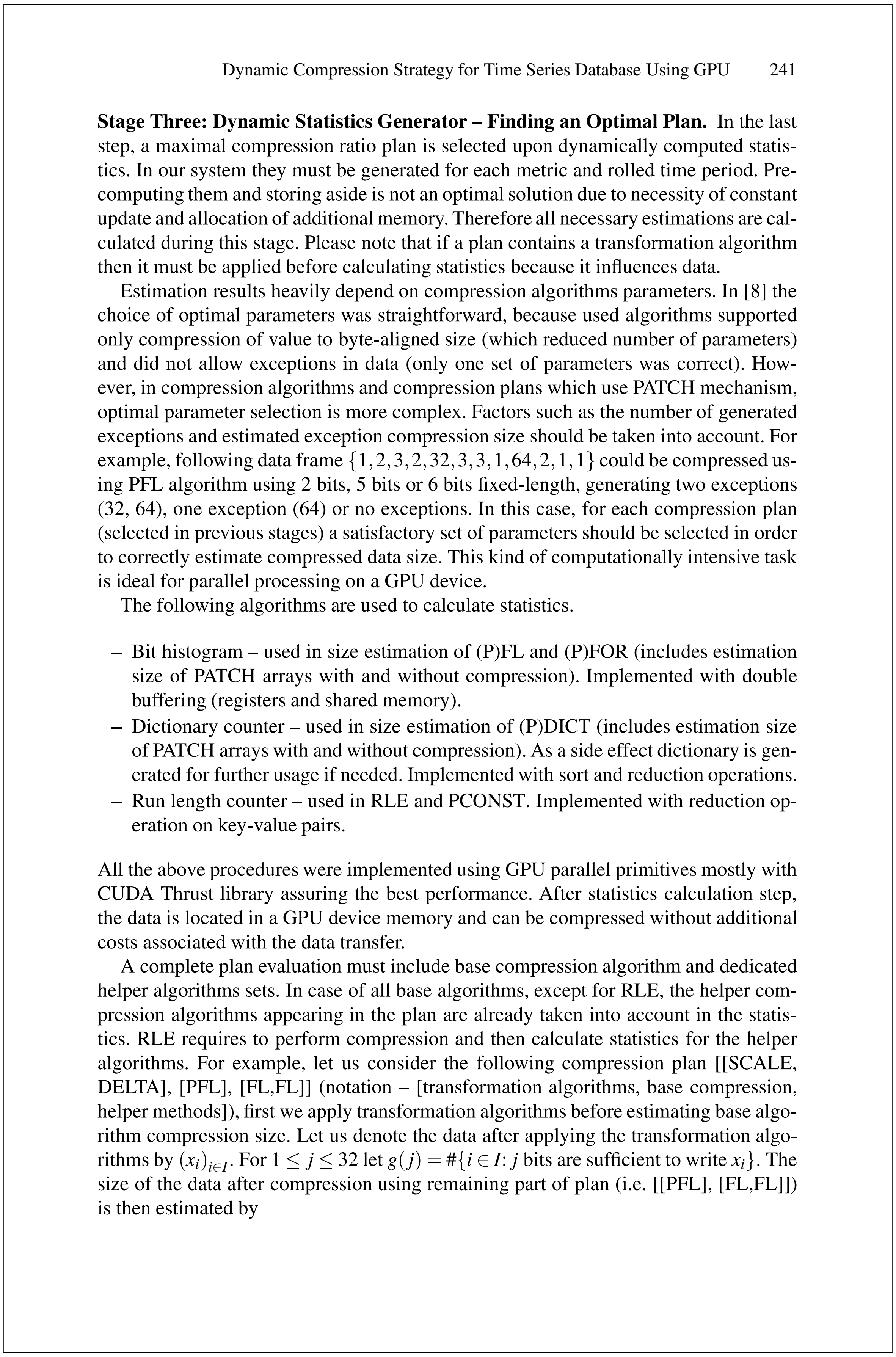
1. **Transformation algorithms (SCALE, DELTA).** All algorithms in this section are optional but may be used together (if present must be applied in the given order). Goal: Improve properties of data storage and prepare for better compression.
2. **Base compression algorithms (PDICT, PFL, PFOR, RLE, PCONST).** Only one algorithm may be selected as the base algorithm. All algorithms in this section use two or more arrays. Some of them, may qualify for further compression using *Helper compression algorithms*.
3. **Helper compression algorithms (FOR, FL, DICT).** The algorithms used to compress selected arrays from the previous step. Each of the resulting arrays can be compressed with only one algorithm. In order to minimize the stages of decompression PATCH algorithms, which could create new arrays for compression, are excluded. The base algorithm used may limit algorithms in this section. For example, exceptions and values arrays in all PATCH algorithms may only be compressed with FL.

Composition of all sensible paths between algorithms in these three categories leaves only 32 suitable compression plans out of former one million. The longest possible cascaded compression plan may be composed of six steps.

**Stage Two: Hints System – Possibility of Manual Tuning.** Another reduction of possible compression plans generated in the first stage can be done manually by a user speeding up further plan choosing. Number and types of hints may vary in different situations. For example, in time series systems timestamps are always sorted and if we consider separated compression methods for timestamps and values we may find different and better plans for them. A hint indicating sorted input may suggest using DELTA before base algorithms. Additionally, for every metric additional features may be specified or even specific compression algorithm may be enforced. Currently supported hints are located in Table 1.

**Table 1.** A sample set of hints for a time series compression planner

| Hints | Meaning |
|---|---|
| SCALE, (P)FL, RLE DELTA, (P)FOR (P)DICT, PCONST | Enforces a specific compression algorithm in the plan. |
| SORTED | Specify whether the data is sorted. |
| TIMESTAMP | Automatically added by system to timestamps. Sets SORTED to True and SCALE to False. |
| DATA | Automatically added by system to time series values. If not otherwise specified sets SORTED to False and SCALE to False. |



**Stage Three: Dynamic Statistics Generator – Finding an Optimal Plan.** In the last step, a maximal compression ratio plan is selected upon dynamically computed statistics. In our system they must be generated for each metric and rolled time period. Precomputing them and storing aside is not an optimal solution due to necessity of constant update and allocation of additional memory. Therefore all necessary estimations are calculated during this stage. Please note that if a plan contains a transformation algorithm then it must be applied before calculating statistics because it influences data.

Estimation results heavily depend on compression algorithms parameters. In [8] the choice of optimal parameters was straightforward, because used algorithms supported only compression of value to byte-aligned size (which reduced number of parameters) and did not allow exceptions in data (only one set of parameters was correct). However, in compression algorithms and compression plans which use PATCH mechanism, optimal parameter selection is more complex. Factors such as the number of generated exceptions and estimated exception compression size should be taken into account. For example, following data frame $\{1,2,3,2,32,3,3,1,64,2,1,1\}$ could be compressed using PFL algorithm using 2 bits, 5 bits or 6 bits fixed-length, generating two exceptions (32, 64), one exception (64) or no exceptions. In this case, for each compression plan (selected in previous stages) a satisfactory set of parameters should be selected in order to correctly estimate compressed data size. This kind of computationally intensive task is ideal for parallel processing on a GPU device.

The following algorithms are used to calculate statistics.

- Bit histogram – used in size estimation of (P)FL and (P)FOR (includes estimation size of PATCH arrays with and without compression). Implemented with double buffering (registers and shared memory).
- Dictionary counter – used in size estimation of (P)DICT (includes estimation size of PATCH arrays with and without compression). As a side effect dictionary is generated for further usage if needed. Implemented with sort and reduction operations.
- Run length counter – used in RLE and PCONST. Implemented with reduction operation on key-value pairs.

All the above procedures were implemented using GPU parallel primitives mostly with CUDA Thrust library assuring the best performance. After statistics calculation step, the data is located in a GPU device memory and can be compressed without additional costs associated with the data transfer.

A complete plan evaluation must include base compression algorithm and dedicated helper algorithms sets. In case of all base algorithms, except for RLE, the helper compression algorithms appearing in the plan are already taken into account in the statistics. RLE requires to perform compression and then calculate statistics for the helper algorithms. For example, let us consider the following compression plan [[SCALE, DELTA], [PFL], [FL,FL]] (notation – [transformation algorithms, base compression, helper methods]), first we apply transformation algorithms before estimating base algorithm compression size. Let us denote the data after applying the transformation algorithms by $(x_i)_{i\in I}$. For $1 \leq j \leq 32$ let $g(j) = \#\{i \in I \colon j \text{ bits are sufficient to write } x_i\}$. The size of the data after compression using remaining part of plan (i.e. [[PFL], [FL,FL]]) is then estimated by

$$E := \min_{1\leq j\leq 32}\left(\sum_{l=1}^{j} g(l)\cdot len(g) + \sum_{l=j+1}^{32} g(l)\left(\lceil \log_2 len(g)\rceil + last(g) - j\right)\right),$$

where $len(g) = \sum_{l=1}^{32} g(l)$ and $last(g) = \max_{1\leq l\leq 32}\{l{:}\, g(l) \neq 0\}$. First sum estimates base algorithm compression size and second estimates compression size of two exception arrays compressed using FL algorithm. If we change PFL to PFOR similar estimation is made but in first step $\min(x_i)_{i\in I}$ is subtracted from all values. PDICT works on dictionary counter array and uses it to build an optimal dictionary with exceptions (i.e. PDICT generate three output arrays and each may be compressed using FL, optimal dictionary with exception is such that minimizes estimated compression size after applying PDICT algorithm and using FL helper algorithm). Detailed description of other evaluation functions is beyond the size limitation of this paper and will be published separately.

## 3 Runtime Results

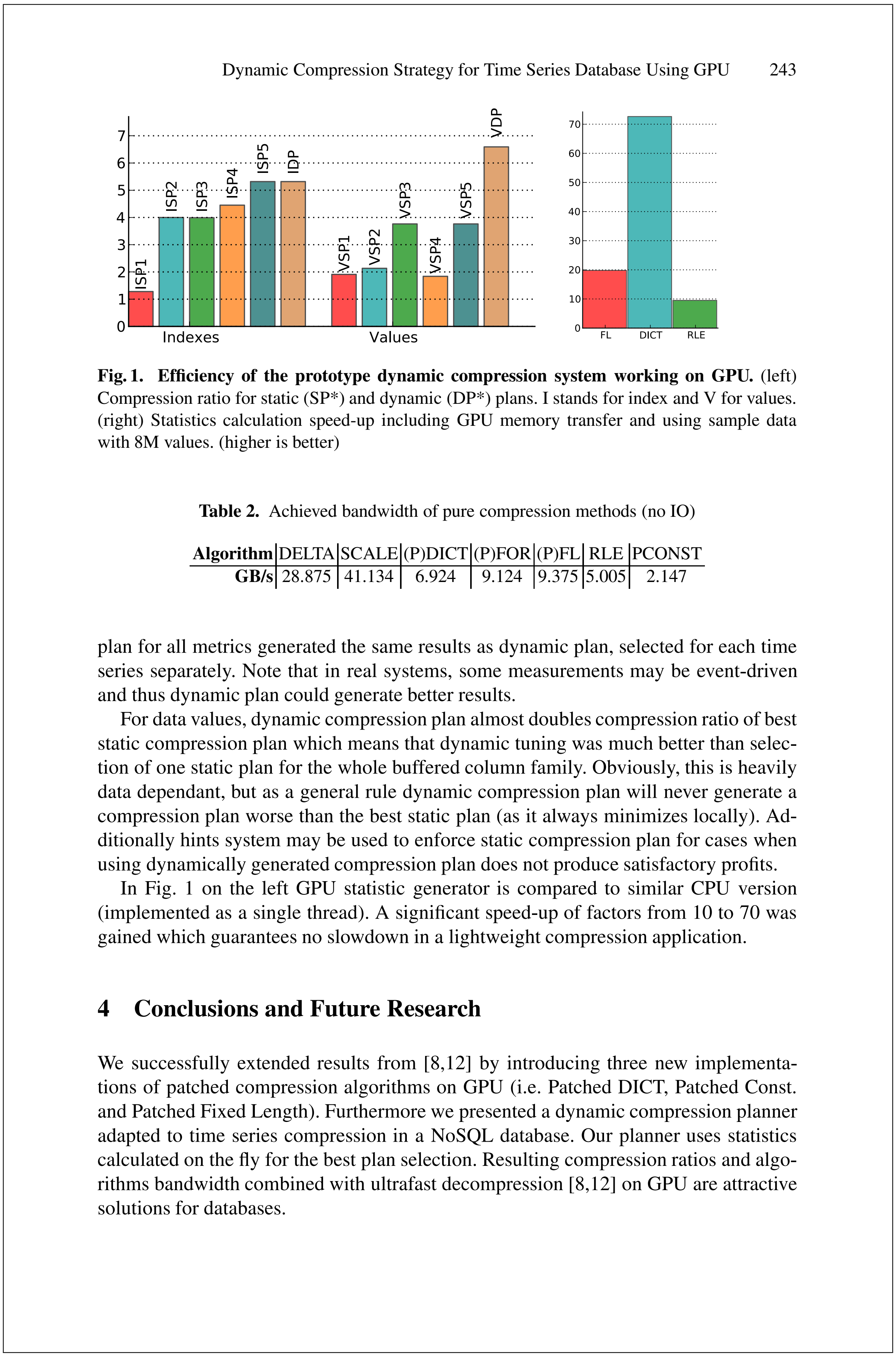
We compared effectiveness of dynamic compression planner and a single static plan within the same CF (Column Family – portion of data rolled in a database) by running the prototype system on samples from a set of network servers monitoring. The data included memory usage, the number of exceptions reported, services occupancy time or CPU load. Data covered a sample of 20 days of constant monitoring and contained about 91K data points in just a few time series (available at `www.mat.umk.pl/~eror/gid2013`). It was taken as a very short and limited sample from a telecommunication monitoring system which collects about 700.000 data points per day. Please note that in this case quality of the sample (its origin) is more important than its length.

We used the following equipment: *Nvidia® Tesla C2070 (CC 2.0)* with 2687 MB; 2 x Six-Core processor *AMD® Opteron™* with 31 GB RAM, *Intel®* RAID Controller *RS2BL040* set in RAID 5, 4 drives *Seagate® Constellation ES ST2000NM0011* 2000 GB, Linux kernel 2.6.38–11 with the CUDA driver version 5.0.

### 3.1 Evaluation of Compression Planer

The evaluation was divided into two parts. The first measured efficiency of dynamic planner and was intended to prove the basic contribution of this work. The second checked efficiency of GPU based statistics evaluation when compared to CPU and proving contribution concerning time efficiency.

Figure 1 on the right shows compression ratio (original size / compressed size) using several static plans (one compression plan for the whole column family) and dynamic plan (dynamically chosen compression plan for different metrics, tags and time ranges). In case of timestamps, five static plans were generated using DELTA algorithm combined with five base compression methods (and helper compression algorithms if suitable). Similarly, for data values five plans where selected except SCALE was used instead of DELTA. We may observe, that for timestamp arrays, compression ratio of dynamic compression plan was equivalent to best static compression plan. This situation appeared because all time series were evenly sampled in this case. Therefore one static



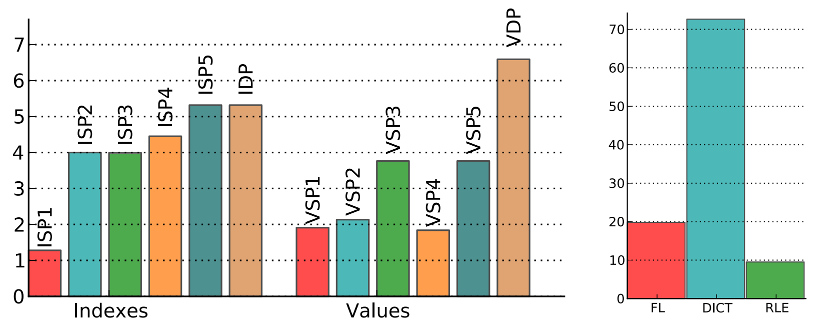


**Fig. 1. Efficiency of the prototype dynamic compression system working on GPU.** (left) Compression ratio for static (SP*) and dynamic (DP*) plans. I stands for index and V for values. (right) Statistics calculation speed-up including GPU memory transfer and using sample data with 8M values. (higher is better)

**Table 2.** Achieved bandwidth of pure compression methods (no IO)

| **Algorithm** | DELTA | SCALE | (P)DICT | (P)FOR | (P)FL | RLE | PCONST |
|---|---|---|---|---|---|---|---|
| **GB/s** | 28.875 | 41.134 | 6.924 | 9.124 | 9.375 | 5.005 | 2.147 |

plan for all metrics generated the same results as dynamic plan, selected for each time series separately. Note that in real systems, some measurements may be event-driven and thus dynamic plan could generate better results.

For data values, dynamic compression plan almost doubles compression ratio of best static compression plan which means that dynamic tuning was much better than selection of one static plan for the whole buffered column family. Obviously, this is heavily data dependant, but as a general rule dynamic compression plan will never generate a compression plan worse than the best static plan (as it always minimizes locally). Additionally hints system may be used to enforce static compression plan for cases when using dynamically generated compression plan does not produce satisfactory profits.

In Fig. 1 on the left GPU statistic generator is compared to similar CPU version (implemented as a single thread). A significant speed-up of factors from 10 to 70 was gained which guarantees no slowdown in a lightweight compression application.

## 4 Conclusions and Future Research

We successfully extended results from [8,12] by introducing three new implementations of patched compression algorithms on GPU (i.e. Patched DICT, Patched Const. and Patched Fixed Length). Furthermore we presented a dynamic compression planner adapted to time series compression in a NoSQL database. Our planner uses statistics calculated on the fly for the best plan selection. Resulting compression ratios and algorithms bandwidth combined with ultrafast decompression [8,12] on GPU are attractive solutions for databases.

Our future work will concentrate on query optimization in hybrid CPU/GPU environment, query execution on partially compressed data and extending dynamic compression planner by introducing additional costs factors (i.e. decompression execution time[8] or potential of query execution on compressed data).

## References

1. Apache HBase (2013), `http://hbase.apache.org`
2. ParStream - website (2013), `https://www.parstream.com`
3. TempoDB – Hosted time series database service (2013), `https://tempo-db.com/`
4. Boncz, P.A., Zukowski, M., Nes, N.: Monetdb/x100: Hyper-pipelining query execution. In: CIDR, pp. 225–237 (2005)
5. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A Distributed Storage System for Structured Data. In: OSDI 2006: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, pp. 205–218 (November 2006)
6. Cloudkick. 4 months with cassandra, a love story (March 2010), `https://www.cloudkick.com/blog/2010/mar/02/4_months_with_cassandra/`
7. Delbru, R., Campinas, S., Samp, K., Tummarello, G.: Adaptive frame of reference for compressing inverted lists. Technical report, DERI – Digital Enterprise Research Institute (December 2010)
8. Fang, W., He, B., Luo, Q.: Database compression on graphics processors. Proceedings of the VLDB Endowment 3(1-2), 670–680 (2010)
9. Fink, E., Gandhi, H.S.: Compression of time series by extracting major extrema. J. Exp. Theor. Artif. Intell. 23(2), 255–270 (2011)
10. Lees, M., Ellen, R., Steffens, M., Brodie, P., Mareels, I., Evans, R.: Information infrastructures for utilities management in the brewing industry. In: Herrero, P., Panetto, H., Meersman, R., Dillon, T. (eds.) OTM 2012 Workshops. LNCS, vol. 7567, pp. 73–77. Springer, Heidelberg (2012)
11. OpenTSDB. Whats opentsdb (2010-2012), `http://opentsdb.net/`
12. Przymus, P., Kaczmarski, K.: Improving efficiency of data intensive applications on GPU using lightweight compression. In: Herrero, P., Panetto, H., Meersman, R., Dillon, T. (eds.) OTM 2012 Workshops. LNCS, vol. 7567, pp. 3–12. Springer, Heidelberg (2012)
13. Przymus, P., Rykaczewski, K., Wiśniewski, R.: Application of wavelets and kernel methods to detection and extraction of behaviours of freshwater mussels. In: Kim, T.-h., Adeli, H., Slezak, D., Sandnes, F.E., Song, X., Chung, K.-i., Arnett, K.P. (eds.) FGIT 2011. LNCS, vol. 7105, pp. 43–54. Springer, Heidelberg (2011)
14. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proc. of the 18th Intern. Conf. on World Wide Web, pp. 401–410. ACM (2009)
15. Zukowski, M., Heman, S., Nes, N., Boncz, P.: Super-scalar ram-cpu cache compression. In: ICDE 2006. Proc. of the 22nd Intern. Conf. on Data Engineering, p. 59. IEEE (2006)

# Time Series Queries Processing with GPU Support*

Piotr Przymus[1] and Krzysztof Kaczmarski[2]

[1] Nicolaus Copernicus University, Poland
eror@umk.mat.pl
[2] Warsaw University of Technology, Poland
k.kaczmarski@mini.pw.edu.pl

**Abstract.** In recent years, an increased interest in processing and exploration of time-series has been observed. Due to the growing volumes of data, extensive studies have been conducted in order to find new and effective methods for storing and processing data. Research has been carried out in different directions, including hardware based solutions or NoSQL databases. We present a prototype query engine based on GPGPU and NoSQL database plus a new model of data storage using lightweight compression. Our solution improves the time series database performance in all aspects and after some modifications can be also extended to general-purpose databases in the future.

**Keywords:** time series database, lightweight compression, data-intensive computations, GPU, CUDA.

## 1 Introduction

Time Series analysis plays a crucial role in many important computational applications. Various hardware or software which must be monitored in order to ensure proper level of service quality emit time series as self describing data. This kind of *machine generated databases* are often growing with square factor since they represent monitoring relations between a distributed system's components in 'all-to-all' fashion. Number of time series generated in this way grows very quickly and falls under category of *Big Data*: problems in which size of data is a problem itself. Classical statistical systems (like R or SAS [7,2]), although capable of performing advanced analysis, are no longer able to handle the newly appearing challenges:

- Very large volumes of data must be consumed by a database in real time. If 1000 machine reports 1000 values every 10 seconds the systems must store $8.64 \cdot 10^9$ data points every day. Because of continuous operation of the system there is no possibility of batch processing.
- Resolution of data cannot be lost. Industrial systems benefit from ability to track single events as well as global tendencies or changes. Correlations between quickly appearing events cannot be found on general level.
- System must be able to answer any kind of queries to the database in reasonable time even if a query involves billions of points. This tight efficiency constrain may be only fulfilled if computation power is not bounded and scales well, possibly linearly.

* The project was funded by National Science Centre, decision DEC-2012/07/D/ST6/02483.

– Storage may not be limited and should scale transparently. SQL databases with centralized indexes are no longer sufficient for these requirements or cannot meet limited budget requirements.

New systems like OpenTSDB [4] or Tempo-DB [6] try to address the above needs by using *Big Table* [8] data model. They are able to import data very efficiently while distributing it in a cloud-like storage. Querying is done by retrieving fragments of data from the distributed regions and putting them together with a map-reduce algorithm. One of the bottlenecks for a time series database is IO bandwidth and centralized aggregation process. Query processing for a longer period of time may need to process hundreds millions of data points. In such cases system reaction time often becomes too long.

### 1.1 General-Purpose Computation on Graphics Processing Units (GPGPU)

GPU programming offers tremendous processing power and excellent scalability with increasing number of parallel threads. However, vector-like processing in GPU has some limitations. One of them is obligatory data transfer between RAM (random-access memory) of the host machine and the computing GPU device, which generates additional cost when compared to a pure CPU-based solution. This barrier can make GPU-based algorithms unsatisfactory especially for smaller problems. One of the goals of this work is to improve efficiency of data transfer between disk, through RAM, global GPU memory and processing unit.

One of the possibilities, and often the only option, to optimize the mentioned data transfer is to reduce its size by compressing. Classical compression algorithms are computationally expensive (gain from the transfer data does not compensate the calculations [18]) and difficult to implement on the GPU [16]. Alternatives such as lightweight compression algorithms which are successfully used for CPU and GPU are therefore very attractive [18,10,12].

This paper addresses optimizations in time series systems like OpenTSDB allowing for faster query response. We present a new model of data storage using lightweight compression and parallel query processing using GPU. The rest of the paper is organized as follows. In the rest of this section we motivate and presents some of the related works concerning time series, big data and GPU processing. In section 2 we explain the prototype system architecture and querying process, while in section 3 a reader may find experimental results. Section 4 concludes.

### 1.2 Motivation

There are evidences of configurations pushing tens of billions data points a day into a monitoring system (like Facebook or Twitter). In such complicated cases system often stores very detailed measurements taken in different metrics and configurations for example every 10 seconds and therefore must deal not only with many points in the same time series but also with a huge number of time series as well.

What we observed in our industrial experience is that a user often performs many different queries working on the same time series in the fixed period of time. The reason for this is that users want to observe the same point in time from many angles, which means performing different types of aggregations of different dimensions. Obviously, this analysis strategy cannot be predicted and aggregations cannot be preprocessed. OpenTSDB saves data points in cache in order not to repeat very expensive hbase scan operation. However, serialized points aggregation was noticed to be slower for large number of time series. Therefore, we propose to use GPU as an alternative query co-processor using our novel lightweight time series compression strategy. What is more important many users already own powerful graphical devices which may be used as local coprocessors for data analysis. Database querying should take into account this new possibility of supporting time consuming computations.

The main motivation of this work is to open new possibilities in time series querying by utilization of GPU processors for ultra fast time series aggregation on both server and client side. In this paper we also show that GPU processor may be used to perform computations on compressed data without introducing any additional costs. This in turn allows for application of GPU processors not only in computation-intensive problems in which time of copying data is amortized by numerical computations but also in data-intensive problems. This achievement opens a new filed for general database algorithms. Our solution improves the overall time series database performance by: minimizing communication footprint between a data storage and a query engine (by using i.a.: data compaction and lightweight compression) and moving data decompression and time series query processing to GPU.

## 1.3 Related Works

There is huge interest in efficient time series processing both in industry and science since large (and growing fast) data sets need to be queried and stored efficiently. OpenTSDB [4] build on top of HBase [1] and offers tremendous speed of data insertion and scanning together with high scalability. However, its data model is limited and so far cannot handle many important cases (like data annotations) Unde et al. [15] claim that OpenTSDB reaches much better performance than DB2 RDBMS for time series processing. Our experiments showed that OpenTSDB performance degrades if there are more than just a few tags attached to single metric which means that it has to aggregate too many time series.

Real time data analytic is offered by ParStream [5] data base system using GPU nodes. Data is equally distributed along machines. Combination of CPU and GPU processing together with load balancing enables to achieve almost real time processing of terabytes of data [11]. Also Jedox [3], an OLAP database system, offers possibility of using GPU as a coprocessor in queries. Both solutions are not strictly focused on time series processing and therefore probably cannot offer many optimizations which could be potentially possible. Our research is aimed at similar goals but with stress on large number of time series.

In [17] authors present interesting study of different compression techniques for WWW data in order to achieve querying speed-up. A general solution for data intensive applications by cache compression is discussed in [18]. Obviously the same technique may be used for time series and the efficiency may be increased if decompression speed is higher than I/O operation. In this paper we also show that decoding is really much faster and eliminates this memory bottleneck. The compression schemes proposed by

Zukowski et al. [18] offer good trade-off between compression time and encoded data size and what is more important are designed especially for super scalar processors which means also very good properties for GPU.

## 2 System Architecture

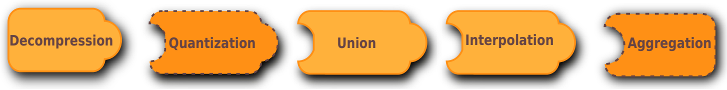
This section presents our system architecture. A set of collectors performs some treatment of source data and send it to the storage. Then data is sent to a storage which can be easily realized by column based NoSql database systems like Hbase [1] or Cassandra [9]. During the insertion process time series are compacted into larger records (taking into account the metric name and tags) containing a specified period of time (eg 15 minutes, 1 hour, 2 hours, 24 hours – depending on the number of observations) which differs from OpenTSDB design. The last important part of the system is the query engine responsible for user-database interactions. Again it must retrieve and process data in time acceptable for a user even if queried time period is long and covers many time series and data points. Following other solutions, as an answer to this problem we propose a analytic coprocessor but with GPU computing support. Since data transfer time is critical for distributed systems the key improvement over any other time series solution is decreasing size of necessary data transfer. In our solution we combine storing data internally compressed with adapted lightweight compression and ultra fast decompression done directly into GPUs memory. This strategy minimises not only storage size but also significantly increases transfer speed and processing time. In the last stage, utilization of GPU allows for very fast query processing.

### 2.1 Query Processing

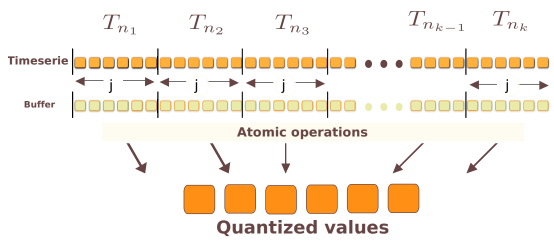
The overall process of query execution is shown in Fig. 1a, while detailed query processing steps are presented below.

**Decompression:** OpenTSDB uses HBase support for lightweight compression algorithms such as Snappy or LZO. However, our observations suggest that the use of specialized lightweight compression algorithms like PFOR and PFOR-DIFF can significantly raise performance. Moreover, lazy decompression can be considered as a one of the stages of query processing, which minimizes the cost of memory transfers. Results are further improved by ultra fast decompression done by GPU processor. Obviously, better compression coefficients can be obtained due to the well-known characterization of the stored data. In this work we use modified PFOR and PFOR-DIFF from our earlier work [12].

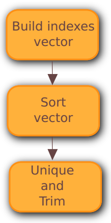
**Quantization:** An important aspect is the analysis of the data at different levels of detail. This means that we have the opportunity to analyse the long-term general aspects as well as short-term detailed ones. Moreover, it allows us to limit the number of details in data, and thus reduce the initial size of it prior to processing. This important part of query processing may be efficiently performed on GPU: each thread examines $j$ data elements (Fig. 1b). In a loop, it makes the quantization of the time series. Quantization is carried out using threads buffers to reduce the number of atomic operations needed for global memory. In the end, the partial results are stored in memory using global atomic operations.



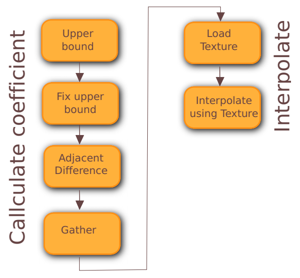
(a) **Query processing steps.** Quantization and Aggregation parts are optional



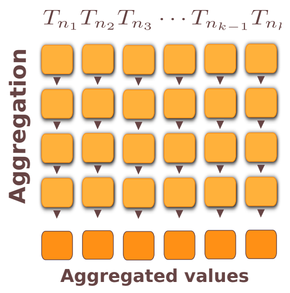
(b) Quantization



(c) Union



(d) Interpolation



(e) Aggregation

**Fig. 1.** Query operations (where $T_{n_1}, T_{n_2}, \ldots, T_{n_{k-1}} T_{n_k}$ are threads)

**Union:** In order to calculate aggregation for non evenly sampled time series, we need to transform them into evenly sampled ones (through interpolation). The first step is to determine a set union of timestamp indices for all time series. Again this stage can be efficiently implemented using *Thrust* GPU library offering basic operations for vectors (the interface is similar to the Standard Template Library vector for C++). In the first step, we build the vector of time series. Then the timestamps are sorted using sort method – which performs highly optimized Radix Sort. Subsequently unique operation is performed which removes duplicate items. See outline in Fig. 1c.

**Interpolation:** In the previous step we calculated the union of timestamp indices ($t_i$). Here, we need to interpolate values for selected timestamps in every time series. Finally, we obtain an evenly sampled time series. To improve efficiency of this part we used textures with CUDA hardware support for linear interpolation. There are also efficient implementations of other types of interpolation [14]. The procedure consists of two parts (see Fig. 1d). First we calculate linear interpolation coefficients, i.e. for each $t$ in the union, we search for $t_i < t < t_{i+1}$ and calculate $t_{i+1} - t_0$. Since the time series are non-uniformly sampled this operation uses vectorized search (upper_bound from *Thrust*). The second step uses a linear interpolation GPU hardware support and uses previously computed factors.

**Aggregation:** Aggregation works on equally sampled time series. For each time point we calculate aggregation across data values in all time series. Each thread in a loop analyses the values of all time series for a single point in time. Then it writes aggregated value to global memory. See Fig. 1e for overview.

## 3 Prototype Query processing

**Query Processing:** The experiments were carried out using a common query for monitoring systems: *Calculate an aggregation of all the time series for a given metric for a specified period of time*. It is a general task which is a starting point for many other analytical methods like finding extreme values, pattern matching or other data mining algorithms. It covers all important aspects of query processing: data retrieval, combination of many time series, missing data and data aggregation.

**Data Settings:** A synthetic set of time series for a single metric with different tags was prepared. It may be treated as one parameter measurement on a set of distributed sensors. The simulated measurements correspond to 600 time series with measurement every 300 seconds with random 10% of elements missing in each time series, which gives approximately $600 \times 16.1K \approx 9.6M$ data points. Additionally, the synthetic data has been prepared to obtain different compression ratios seen in real applications [13].

**Environment Settings:** Experiments were carried out on one instance of HBase, query processing was conducted on the database server. Hardware configuration: Two six core processors *Intel® Xeon® E5649 2.53GHz*, 8GB RAM and *Nvidia® Tesla M2070* card. Tests were carried out using 600 time-series containing from $2.0K$ to $16.1K$ of observations, average processing time for 25 launches was taken. Because processed data in most cases fit in the HBase cache, configuration with LZO (Lempel-Ziv-Oberhumer) compression achieves only slightly better results than with no compression. In industrial applications, queries are less likely to hit the cache and the acceleration of LZO compression is higher.

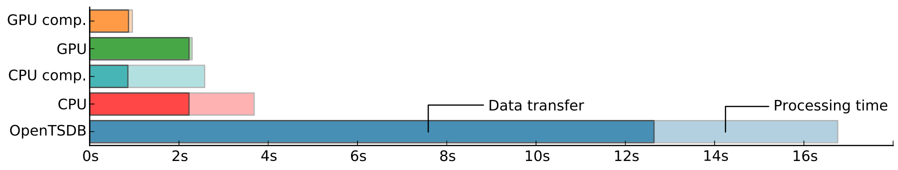
**OpenTSDB:** A modified version of a console client (modified to log query execution time after doing a warm-up phase of Java virtual machine) and Hbase configured with LZO compression were used in experiments.

**Prototype:** Developed using C++ and CUDA C++ and Thrift protocol. HBase was configured without compression, instead highly tuned lightweight (de)compression algorithms for time series where used.

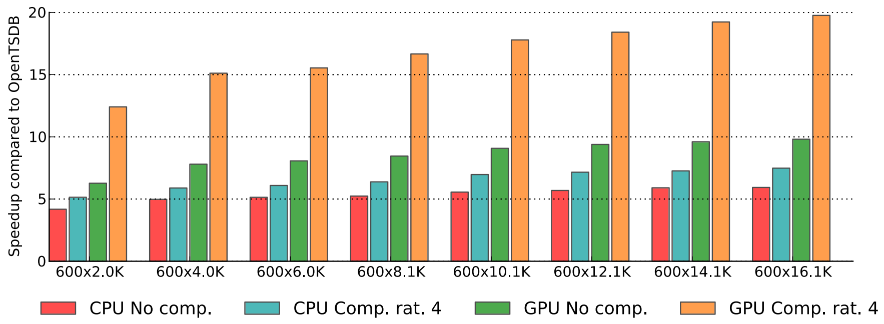
### 3.1 Results and Discussion

The comparison of OpenTSDB and our prototype performance is eligible due to similar architecture of both solutions and identical query processing work-flow. All differences are discussed bellow. The following factors were considered: the data transfer time (the time between sending a query to HBase and receiving the results) as well as the time needed to process the data (including data decompression), the time required to exchange data with the GPU (if used) and the processing time.

A detailed timeline for query execution with fixed data size (600 time series $\times 16.1K$ observations) is provided in Figure 2a. We can observe that processing in OpenTSDB



(a) **Execution time** 600 time-series, 16.1*K* observation each (lower is better)



(b) **Prototype speedup compared to OpenTSDB** (higher is better)

**Fig. 2.** Measured results

takes only 25% of query time and despite being 2.8 times slower than CPU prototype, it is not the main bottleneck. Better performance is not just a matter of changing Java to C++. It is *data compaction* to reduce processed data size and number of fetched rows and columns and increase efficiency of data transfer. Using data compaction, data transfer performance significantly increases (5.7 times faster) for both prototypes (CPU an GPU). But still most of the time is spent on communication with the database. What is more GPU is 21x faster than CPU when comparing data processing speed. Thus calculations are limited by the data transfer. It is therefore necessary to improve data transfer in order to achieve better results. This is done by using efficient lightweight compression implementation [12]. Lightweight compression introduces only a slight improvement in CPU prototype. This is because of the relatively long time needed for the data processing (almost 1/4 of total time – see *CPU* in Fig. 2a). This is because the lightweight compression significantly reduces data transfer time, but it also increases the data processing time (see *CPU comp.* in Fig. 2a). Computation and communication with the GPU is only a small fraction of the entire query and decompression adds only a small overhead to the query (see GPU and GPU comp. in Fig. 2a).

Figure 2b presents the resulting acceleration obtained on CPU and GPU prototype (in comparison to OpenTSDB query) on different data sizes. Both figures include CPU and GPU prototypes with and without lightweight compression. Due to the page limit only results for one compression ratio (4) are presented. Notice that the size of the data is important and better results can be obtained on larger data sets. It is also worth of noting that the data transfer is often a bottleneck in many GPGPU applications. This was also the case, however, through the use of a lightweight compression, data transfer is highly improved, thereby significantly speeding up the execution of the query.

## 4 Conclusions and Future Work

Time series databases play a crucial role in many branches of industry. Machine generated measurements require fast, real-time insertion and almost real-time querying. We showed that in case of computations dedicated to time series the existing solutions may be improved by utilization of GPU processors. So far data intensive application had to overcome the problem of additional CPU to GPU data transfer cost. Only algorithms of more than linear computation time complexity could benefit from parallel GPU processing. In this paper we showed that by introduction of fine tuned compression methods we can improve these results. Especially time series processing may speed-up significantly when compared to industrial solutions or experimental CPU prototypes.

Our future work will concentrate on query optimization in hybrid CPU/GPU environment, query execution on partially compressed data and on developing dynamic compression planer.

## References

1. Apache HBase (2013), `http://hbase.apache.org`
2. Business Intelligence and Analytics Software - SAS (2013), `http://www.sas.com/`
3. Jedox - website (2013), `https://www.jedox.com`
4. OpenTSDB - A Distributed, Scalable Monitoring System (2013), `http://opentsdb.net/`
5. ParStream - website (2013), `https://www.parstream.com`
6. TempoDB – Hosted time series database service (2013), `https://tempo-db.com/`
7. The R Project for Statistical Computing (2013), `http://www.r-project.org/`
8. Chang, F., et al.: Bigtable: A Distributed Storage System for Structured Data. In: OSDI 2006: Seventh Symposium on Operating System Design and Implementation, pp. 205–218 (2006)
9. Cloudkick. 4 months with cassandra, a love story (March 2010), `https://www.cloudkick.com/blog/2010/mar/02/4_months_with_cassandra/`
10. Fang, W., He, B., Luo, Q.: Database compression on graphics processors. Proceedings of the VLDB Endowment 3(1-2), 670–680 (2010)
11. ParStream. ParStream - Turning Data Into Knowledge - White Paper. Technical report (2010)
12. Przymus, P., Kaczmarski, K.: Improving efficiency of data intensive applications on GPU using lightweight compression. In: Herrero, P., Panetto, H., Meersman, R., Dillon, T. (eds.) OTM-WS 2012. LNCS, vol. 7567, pp. 3–12. Springer, Heidelberg (2012)
13. Przymus, P., Rykaczewski, K., Wiśniewski, R.: Application of wavelets and kernel methods to detection and extraction of behaviours of freshwater mussels. In: Kim, T.-h., Adeli, H., Slezak, D., Sandnes, F.E., Song, X., Chung, K.-i., Arnett, K.P. (eds.) FGIT 2011. LNCS, vol. 7105, pp. 43–54. Springer, Heidelberg (2011)
14. Ruijters, D., ter Haar Romeny, B.M., Suetens, P.: Efficient gpu-based texture interpolation using uniform b-splines. Journal of Graphics, GPU, and Game Tools 13(4), 61–69 (2008)
15. Unde, P., et al.: Architecting the database access for a it infrastructure and data center monitoring tool. In: ICDE Workshops, pp. 351–354. IEEE Computer Society (2012)
16. Wu, L., Storus, M., Cross, D.: Cs315a: Final project cuda wuda shuda: Cuda compression project. Technical report, Stanford University (March 2009)
17. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proc. of the 18th Intern. Conf. on World Wide Web, pp. 401–410. ACM (2009)
18. Zukowski, M., Heman, S., Nes, N., Boncz, P.: Super-scalar ram-cpu cache compression. In: ICDE 2006, Proc. of the 22nd intern. conf. on Data Engineering, pp. 59–59. IEEE (2006)

# A Bi-objective Optimization Framework for Query Plans

Piotr Przymus[1], Krzysztof Kaczmarski[2], and Krzysztof Stencel[3]

[1] Nicolaus Copernicus University, Poland, eror@mat.umk.pl
[2] Warsaw University of Technology, Poland, k.kaczmarski@mini.pw.edu.pl
[3] The University of Warsaw, Poland, stencel@mimuw.edu.pl

**Abstract.** Graphics Processing Units (GPU) have significantly more applications than just rendering images. They are also used in general-purpose computing to solve problems that can benefit from massive parallel processing. However, there are tasks that either hardly suit GPU or fit GPU only partially. The latter class is the focus of this paper. We elaborate on hybrid CPU/GPU computation and build optimisation methods that seek the equilibrium between these two computation platforms. The method is based on heuristic search for bi-objective Pareto optimal execution plans in presence of multiple concurrent queries. The underlying model mimics the commodity market where devices are producers and queries are consumers. The value of resources of computing devices is controlled by supply-and-demand laws. Our model of the optimization criteria allows finding solutions of problems not yet addressed in heterogeneous query processing. Furthermore, it also offers lower time complexity and higher accuracy than other methods.

## 1 Introduction

General-Purpose computing on Graphics Processing Units (GPGPU) involves utilization of graphics processing units (GPU) in tasks traditionally handled by central processing units (CPU). GPUs offer a notable processing power for streams.

Execution of database queries is an example of a successful application of GPGPU. The current research focuses on using the GPU as a co-processor [5]. GPU as co-processor may accelerate numerous database computations, e.g. relational query processing, query optimization, database compression or supporting time series databases [5,13,14].

An application of GPU requires transferring data from the CPU memory to the graphical device memory. The data transfer is usually time-consuming. It may diminish the gain of the acceleration credited to GPU. This situation can be improved by using lightweight compression methods that can significantly reduce the costs associated with communication [13,14]. However, this does not solve all the problems. In particular, GPU is optimized for numerical computation. Thus, only selected operations will benefit from GPU. Small data sets are another problem. For such sets the data transfer may dominate processing time and destroy the performance gain. Therefore, joint processing capabilities of both CPU and GPU are worth considering. Furthermore, as it is common to have more than one GPU in a computer, a potential use of various GPU devices should be considered. This type of query plans is called heterogeneous.

The previous research efforts focused on the creation of query plans based on a cost model. This approach finds plans with the best throughput. However, it does not allow modelling all phenomena that can occur in heterogeneous systems. Performing a query as soon as possible is not always cost effective [8]. For this reason, we propose a query processing model based on concepts of markets that are known to be suitable for describing the interactions in a heterogeneous world. They have already gained a considerable interest in the context of task processing in heterogeneous systems [6]. In market models, manufacturers (processing devices) compete with each other for customers (query plans). Similar competition occurs among customers.

In this paper, we propose a query optimization model based on the commodity market. A query plan is bi-objectively optimized to minimize: the processing time and the value of consumed resources. For the user, a small difference in execution time can be negligible. Thus, it is worth optimizing a query, so that the execution time satisfies the user while other costs are minimized. In this case, the cost may be, e.g. the responsiveness of the system, power consumption, heat production, etc. One can also consider expressing the cost in financial terms.

## 2 Preliminaries

### 2.1 GPU and Heterogeneous query processing

From the parallel processing's point of view, CPU accompanied by a GPU co-processor is a *shared nothing architecture*. A GPU card has its own memory or a separate area in the CPU main memory. Thus, the data has to be explicitly transferred from the CPU main memory to the GPU main memory. Similarly, the results produced by GPU have to be transferred back to the CPU main memory. This data transfer often introduces significant overhead. Thus, it is important to include the transfer cost in the total execution time of an operation. This cost is also a component of the execution time prediction.

Contemporary computer systems often include more than one GPU. Then, it is possible to combine multiple computational units in a single query plan. Such plans are called *heterogeneous query processing*. Each device may have a different communication cost (e.g. PCIe or shared memory) with the CPU main memory. Furthermore, devices can often communicate directly between each other. Therefore, the main problem of heterogeneous query processing is the construction of such a query plan that uses only computational units from which query performance will benefit most and yet will minimize used resources.

Bress et. al. [5] identified problems of hybrid (CPU/GPU) query processing which are also true in heterogeneous query processing:

**Problem 1** Execution Time Prediction - as multiple database operations may be executed concurrently it is hard to predict influence of concurrent tasks on execution times.

**Problem 2** Critical Query - since the GPU memory, the concurrent GPU kernels execution and the PCIe bus bandwidth are all limited, only the *critical queries* should be selected to use GPU (i.e., queries that benefit from GPU usage and are *important* from global perspective).

**Problem 3** Optimization Impact - as concurrent heterogeneous queries will influence each other, it is important to consider this aspect in the planning process.

### 2.2 Commodity market approach in query processing context

In this paper we address these problems by showing that they may be solved by applying a supply-and-demand pricing model taken from a commodity market. In such a market resource owners (processing devices) price their assets and charge their customers (queries) for consumed resources. Other pricing models may also be used [6].

In the supply-and-demand model when supply (available resources) or demand (needed resources) changes, the prices will be changed until an equilibrium between supply and demand is found. Typically the value of a resource is influenced by: its strength, physical cost, service overhead, demand and preferences [6]. A consumer may be charged for various resources like CPU cycles, memory used, the bus usage or the network usage. Typically, a broker mediates between the resource owners and the consumer. The resource owners announce their valuation and the resource quality information (e.g. estimated time) in response to the broker's enquiry. Then, the broker selects resources that meet the consumer utility function and objectives, like cost and estimated time constraints or minimization of one of the objectives.

### 2.3 Bi-objective optimization

Bi-objective optimization is a problem where optimal decisions need to be taken in the presence of trade-offs between two conflicting objectives. It is a special case of multiple criteria decision making. Typically there are no solutions that meets all objectives. Thus, a definition of an optimum solution set should be established. In this paper we use the predominant Pareto optimality [11]. Given a set of choices and a way of valuing them, the Pareto set consists of choices that are Pareto efficient. A set of choices is said to be Pareto efficient if we cannot find a reallocation of those choices such that the value of a single choice is improved without worsening values of others choices. As bi-objective query optimization is NP-hard, we need an approximate solution [12].

## 3 Heterogeneous Query Planer

The main aim of the new query planner is to propose a solution to the problems listed in Section 2.1, i.e., Execution Time Prediction, Critical Query and

Optimization Impact. Furthermore, this planner also addresses heterogeneous GPU cards and distributed processing. In this paper we propose a method to build heterogeneous query plans based on the economics of commodity markets. It is characterized by the fact that the resource producers determine the cost of their resources and resource consumers jostle for resources. Furthermore, the resources owners provide information on the quality of their resources, i.e., the estimated processing time.

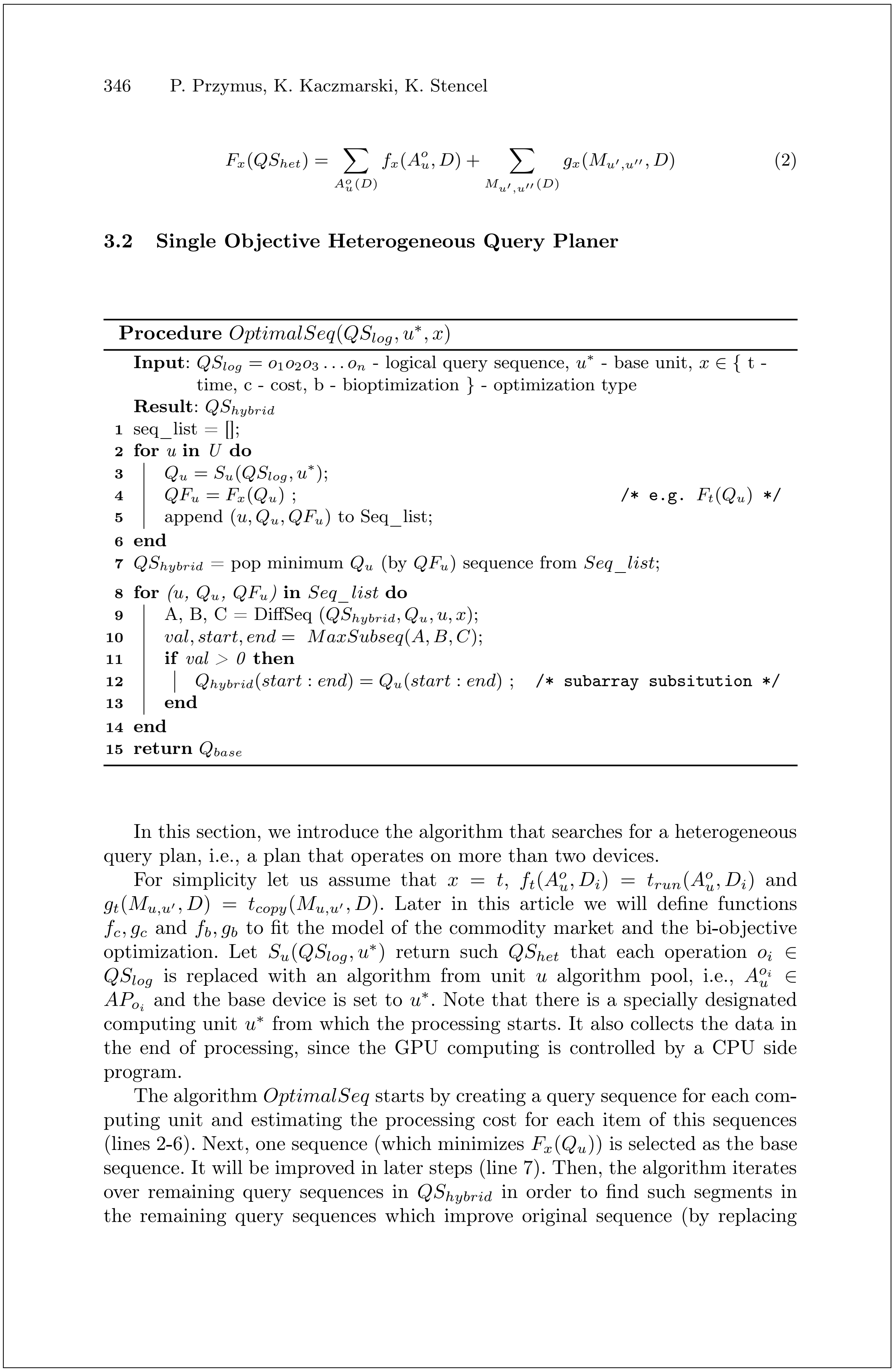
## 3.1 Notation

Table 1 contains a summary of the notation used in this paper. Assume a set of units $U$, a logical query sequence $QS_{log}$ and a dataset $D$. The goal is to build a heterogeneous query sequence. Let $QS_{het}$ be a heterogeneous query sequence defined in Equation (1). Let $D_{k+1}$ be the data returned by an operation $A^{o_k}_{u_{i_k}}(D_k)$. The first row of $QS_{het}$ is created by replacing each operation $o_i \in QS_{log}$ with an algorithm $A^{o_i}_{u_j} \in AP_{o_i}$. The second row is created by inserting an operation $M_{u_k,u_{k'}}(D)$ that copies the output of an algorithm on the unit $u$ to the input of the current algorithm on the unit $u'$.

| **Symbol** | **Description** |
|---|---|
| $U = \{u_1, u_2, \dots u_n\}$ | set of computational units available to process data |
| $D$ | dataset |
| $M_{u_k,u_{k'}}(D)$ | **if** $u_k \neq u_{k'}$ move $D$ from $u_k$ to $u_{k'}$ **else** pass |
| $o_i \in O$ | database operation $o_i$ from set of operations $O$ |
| $A^{o_i}_{u_k}$ | algorithm that computes the operation $o_i$ on $u_k$ |
| $AP_{o_i} = \{A^{o_i}_{u_1}, A^{o_i}_{u_2}, \dots, A^{o_i}_{u_n}\}$ | algorithm pool for the operation $o_i$ |
| $t_{run}(A^{o_i}_{u_j}, D)$ | estimated run time of the algorithm $A^{o_i}_{u_j}$ on the data $D$ |
| $t_{copy}(M_{u_i,u_j}, D))$ | estimated copy time of the data $D$ from $u_i$ to $u_j$ |
| $c_{run}(A^{o_i}_{u_j}, D)$ | estimated run cost of the algorithm $A^{o_i}_{u_j}$ on the data $D$ |
| $c_{copy}(M_{u_i,u_j}, D)$ | estimated copy cost of the data $D$ from $u_i$ to $u_j$ |
| $QS_{log} = o_1 o_2 \dots o_n$ | logical query sequence |
| $QS_{het}$ | heterogeneous query sequence see Eq. 1 |
| $f_t, g_t$ | estimated algorithm run time and copy time see Sec. 3.2 |
| $f_c, g_c$ | estimated algorithm run cost and copy cost see Sec. 3.3 |
| $f_b, g_b$ | estimated algorithm run and copy bi-objective scalarization see Sec. 3.4 |
| $F_x(QS_{het})$ | sum of $f_x$ and $g_x$ over columns of $QS_{het}$ where $x \in \{t, c, b\}$ see Eq. 2 |

Table 1: Symbols used in the definition of our optimisation model

$$QS_{het} = \begin{pmatrix} A^{o_1}_{u_{i_1}}(D_1), & A^{o_2}_{u_{i_2}}(D_2), & A^{o_3}_{u_{i_3}}(D_3), & \dots, & A^{o_n}_{u_{i_n}}(D_n) \\ M_{u^*,u_{i_1}}(D_1), & M_{u_{i_1},u_{i_2}}(D_2), & M_{u_{i_2},u_{i_3}}(D_3), & \dots, & M_{u_{i_n},u^*}(D_n) \end{pmatrix} \quad (1)$$



$$F_x(QS_{het}) = \sum_{A_u^o(D)} f_x(A_u^o, D) + \sum_{M_{u',u''}(D)} g_x(M_{u',u''}, D) \tag{2}$$

### 3.2 Single Objective Heterogeneous Query Planer

**Procedure** *OptimalSeq*($QS_{log}, u^*, x$)

```
Input: $QS_{log} = o_1 o_2 o_3 \ldots o_n$ - logical query sequence, $u^*$ - base unit, $x \in$ { t -
       time, c - cost, b - bioptimization } - optimization type
Result: $QS_{hybrid}$
1  seq_list = [];
2  for $u$ in $U$ do
3      $Q_u = S_u(QS_{log}, u^*)$;
4      $QF_u = F_x(Q_u)$ ;                                   /* e.g. $F_t(Q_u)$ */
5      append $(u, Q_u, QF_u)$ to Seq_list;
6  end
7  $QS_{hybrid}$ = pop minimum $Q_u$ (by $QF_u$) sequence from $Seq\_list$;
8  for ($u$, $Q_u$, $QF_u$) in $Seq\_list$ do
9      A, B, C = DiffSeq ($QS_{hybrid}, Q_u, u, x$);
10     $val, start, end = MaxSubseq(A, B, C)$;
11     if $val > 0$ then
12         $Q_{hybrid}(start : end) = Q_u(start : end)$ ;   /* subarray subsitution */
13     end
14 end
15 return $Q_{base}$
```

In this section, we introduce the algorithm that searches for a heterogeneous query plan, i.e., a plan that operates on more than two devices.

For simplicity let us assume that $x = t$, $f_t(A_u^o, D_i) = t_{run}(A_u^o, D_i)$ and $g_t(M_{u,u'}, D) = t_{copy}(M_{u,u'}, D)$. Later in this article we will define functions $f_c, g_c$ and $f_b, g_b$ to fit the model of the commodity market and the bi-objective optimization. Let $S_u(QS_{log}, u^*)$ return such $QS_{het}$ that each operation $o_i \in QS_{log}$ is replaced with an algorithm from unit $u$ algorithm pool, i.e., $A_u^{o_i} \in AP_{o_i}$ and the base device is set to $u^*$. Note that there is a specially designated computing unit $u^*$ from which the processing starts. It also collects the data in the end of processing, since the GPU computing is controlled by a CPU side program.

The algorithm *OptimalSeq* starts by creating a query sequence for each computing unit and estimating the processing cost for each item of this sequences (lines 2-6). Next, one sequence (which minimizes $F_x(Q_u)$) is selected as the base sequence. It will be improved in later steps (line 7). Then, the algorithm iterates over remaining query sequences in $QS_{hybrid}$ in order to find such segments in the remaining query sequences which improve original sequence (by replacing

```
Procedure DiffSeq($seq_{base}, seq_u, u, x$)
  Input: $seq_{base}$ - base sequence, $seq_u$ - unit query sequence, $u$ - $seq_u$ unit, $x \in \{$
         t - time, c - cost, b - bioptimization $\}$ - optimization type
  Result: A - operations improvement array; B, C - copy to/from unit arrays
1 A, B, C = [], [], [];
2 for $i$ in enumerate columns $seq_{base}$ do
3     $A^o_{u_b}(D_i), M_{u_f,u_t}(D_i) = seq_{base}[i]$;
4     $A^o_u(D_i), M_{u,u}(D_i) = seq_u[i]$;
5     append $f_x(A^o_{u_b}, D_i) - f_x(A^o_u, D_i)$ to A ;        /* e.g. $f_t(A^o_u, D)$ */
6     append $g_x(M_{u_f,u}, D_i)$ to B ;                       /* e.g. $g_t(M_{u,u'}, D)$ */
7     append $g_x(M_{u,u_t}, D_i)$ to C;
8 end
9 return $A, B, C$
```

corresponding segment of the original sequence). This is done by calculating the improvement and copy cost arrays in DiffSeq and finding maximal sequence segment in MaxSubseq. A following variant of the proposed algorithm should also be considered. Suppose that only one query sequence segment may be inserted (i.e., choose one sequence segment from remaining $k-1$ sequences with the biggest), this minimizes number of involved computational units and reduces overall communication costs.

The procedure DiffSeq simply calculates element wise difference between two query sequences $f_x(A^o_{u_b}, D_i) - f_x(A^o_u, D_i)$ and copy costs from/to unit. The procedure MaxSubseq is based on Kadane's algorithm for maximum subarray problem [1]. It scans through the improvement array, computing at each position the maximum subsequence ending at this position. This subsequence is either empty or consists of one more element than the maximum subsequence ending at the previous position. Additionally, the *copy to* and *copy from* costs are included in the calculation of the maximum subsequence ($B$ and $C$ arrays). The algorithm returns the maximum improvement for a subsequence (which may be zero if the subsequence does not improve the original query), the start and end items of subsequence.

The complexity of *OptimalSeq* is $O(k * n)$ where $k$ is the number of devices (usually small) and $n$ is the number of operations of the sequence. $S_u$,$F_x$, DiffSeq and MaxSubseq have the complexity $O(n)$.

## 3.3 Economics in Heterogeneous Environment

To cope with the problems mentioned in Section 2.1, additional criteria are necessary – in this work an approach based on a simple economic model is proposed. Each consumer (client) has a query budget that can be used to pay for the resources used to process queries. Each computational unit is a service provider (producer) of services available in units algorithm pool $AP_{u_i}$. Each service provider establishes its own pricing for execution of any service from $AP_{u_i}$. Pricing of the service depends on:

```
Procedure MaxSubseq(A, B, C)
Input: A - operations improvement array; B, C - copy to/from unit arrays
Result: maximum_improvment, start, end
max_ending_here = max_so_far = 0 ;
begin = tbegin = end = 0 ;
for i, x in enumerate(A) do
    max_ending_here = max(0, max_ending_here + x) ;
    if max_ending_here = 0 then
        tbegin = i ;
        max_ending_here -= B[i];
    end
    if max_ending_here - C[i] >= max_so_far then
        begin = tbegin ;
        end = i ;
        max_so_far = max_ending_here ;
    end
end
return max_so_far - C[end], begin, end
```

– the estimation of needed resources (the size of the data $D$, the performance of the task $A^{o_i}_{u_i}$),
– pricing of needed resources (the load of device $u_i$ – the greater the load on the device, the higher cost of using the device),
– the preference of the device (e.g. device may prefer larger jobs and/or tasks that give a greater acceleration on the GPU).

First, pricing for using the resources of computational unit is established. This depends on the previous load of the device: the higher demand for computational unit, the higher price for using it. This is a periodic process which calculates prices every $\varDelta t_{up}$ seconds by calculating computational unit price $P_u$. Let $0 < L_{curr} < 1$, $0 < L_{prev} < 1$ be current and previous computational unit load factors. Additionally, let $L_{th}$ be a threshold below which prices should decrease, and $P_{min}$ be the minimal price. Then the price is calculated using the following formula[4]:

$$P_u := \begin{cases} max(P_{min}, P_u \cdot (1 + \frac{\varDelta P_u}{(1-\varDelta U)}) & \text{if } (\varDelta P > 0 \wedge \varDelta U > 0) \vee (\varDelta P < 0), \\ P_u & \text{otherwise,} \end{cases} \tag{3}$$

where $\varDelta P_u = L_{current} - L_{threshold}$ and $\varDelta U_u = L_{current} - L_{Previous}$. This is similar to the dynamic pricing model proposed in [16] with exception to the pricing formula i.e., we use $max(P_{min}, P_u \cdot (1 + \frac{\varDelta P_u}{(1-\varDelta U)})$ instead of $max(P_{min}, P_u \cdot (1 + \varDelta P_u))$, this modification reduces the excessive growth of prices.

To reflect the preference of the device in price we need to define a function returning speedup factor between base device $u^*$ (defined in the previous section)

[4]Slightly abusing notation we will also denote the new price by $P_u$.

and current device: $speedup(A_u^o, D_i) = t_{run}(A_{u^*}^o, D_i)/t_{run}(A_u^o, D_i)$. Then we define a cost function as $c_{run}(A_u^o, D_i) = \frac{\#D_i}{speedup(A_u^o,D_i)} \cdot P_u$, where $\frac{\#D_i}{speedup(A_u^o,D_i)}$ part combines the estimation of needed resources and the preference of the device. A computational unit with high speedup on given operation will get a discount per data size when pricing this operation. Similarly, operations with a lower speedup factor will be charged more per quantity. Additionally it is observed [13] that often speedup depends on the size of processed data (usually low speed-up on small datasets) so discount depends on data size.

It is also important to include cost of data transfer, let us define it as

$$c_{copy}(M_{u,u'}, D) := \begin{cases} 0 & \text{if u,u' share memory} \\ \frac{\#D_i}{bandwidth(\#D_i,u,u')} \cdot (P_u + P_{u'})/2 & \text{otherwise} \end{cases}$$

where *bandwidth* returns estimated bytes per second between $u$ and $u'$ computational units. If direct data transfer is not available between $u$ and $u'$ devices, then transit device will be used (e.q. two GPU cards without direct memory access will communicate using CPU RAM).

Now let $f_c(A_u^o, D_i) = c_{run}(A_u^o, D_i)$ and $g_c(M_{u,u'}, D) = c_{copy}(M_{u,u'}, D)$. A solution minimizing the cost may be found under the previous assumptions and using procedure *OptimalSeq*.

### 3.4 Bi-objective Heterogeneous Query Planer

As finding Pareto optimal bi-objective query plan is NP-hard (bi-objective shortest path problem) [12], we will use previously described *OptimalSeq* single objective approximation algorithm and extend it to bi-objective case.

We will use *a priori* articulation of preference approach which is often applied to multi-objective optimization problems. It may be realized as the scalarization of objectives, i.e., all objective functions are combined to form a single function. In this work we will use *weighted product* method, where weights express user preference [11]. Let us define:

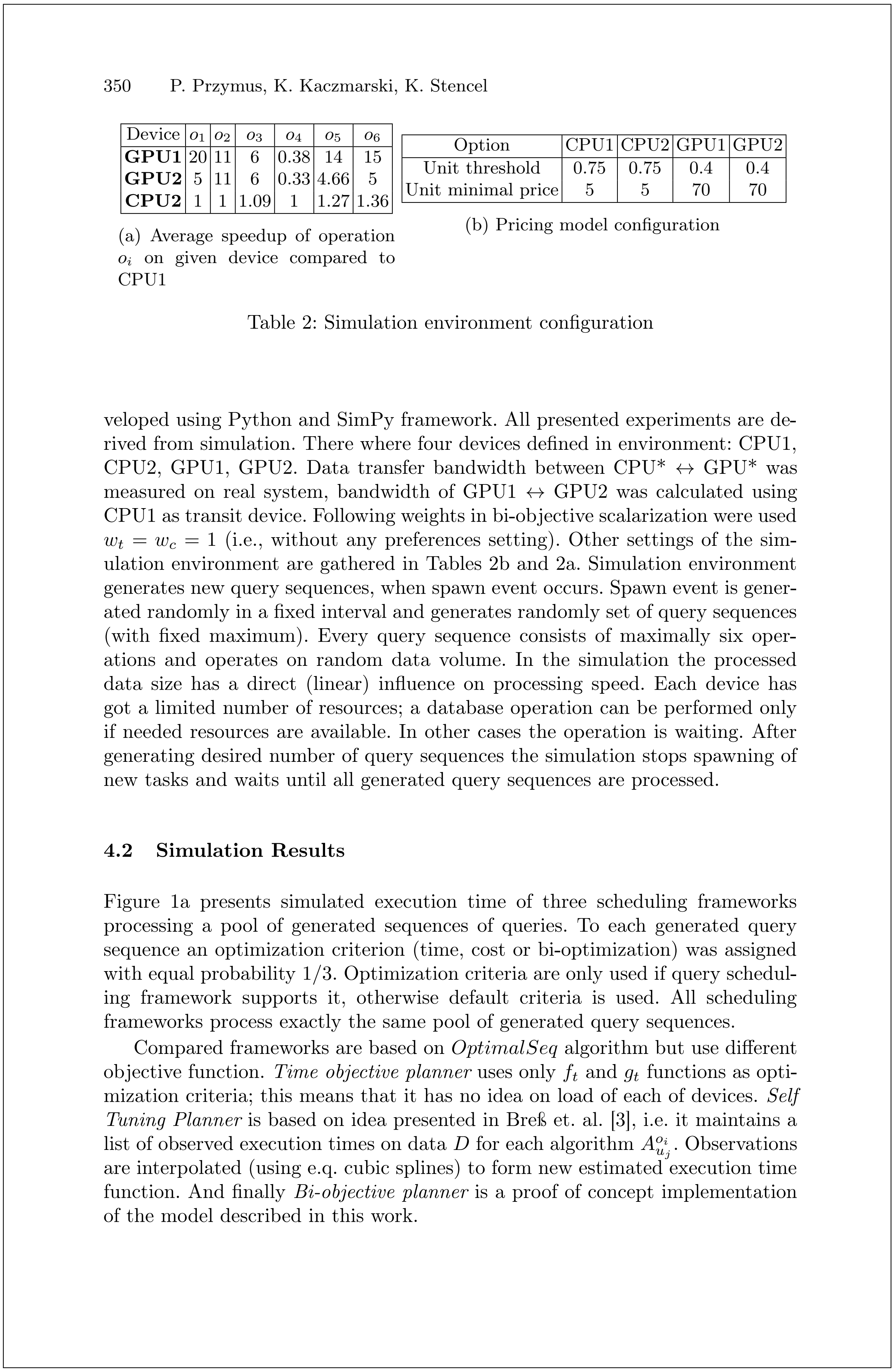
$$f_b(A_u^o, D) = c_{run}(A_u^o, D)^{w_c} \cdot t_{run}(A_u^o, D)^{w_t},$$
$$g_b(M_{u,u'}, D) = c_{copy}(M_{u,u'}, D)^{w_c} \cdot t_{copy}(M_{u,u'}, D)^{w_t}.$$

where $w_t$ and $w_c$ are weights which reflect how important cost and time is (the bigger the weight the more important the feature – values of $f_b, g_b$ are higher than 1). It is worth to mention that a special case with $w_t = w_c = 1$ (i.e., without any preferences) is equivalent to Nash arbitration method (or objective product method) [11].

## 4 Preliminary Experimental Results

### 4.1 Simulation settings

In order to evaluate this model we prepared a *proof of concept* and evaluated it using custom developed simulation environment. Simulation environment was de-



| Device | $o_1$ | $o_2$ | $o_3$ | $o_4$ | $o_5$ | $o_6$ |
|---|---|---|---|---|---|---|
| **GPU1** | 20 | 11 | 6 | 0.38 | 14 | 15 |
| **GPU2** | 5 | 11 | 6 | 0.33 | 4.66 | 5 |
| **CPU2** | 1 | 1 | 1.09 | 1 | 1.27 | 1.36 |

(a) Average speedup of operation $o_i$ on given device compared to CPU1

| Option | CPU1 | CPU2 | GPU1 | GPU2 |
|---|---|---|---|---|
| Unit threshold | 0.75 | 0.75 | 0.4 | 0.4 |
| Unit minimal price | 5 | 5 | 70 | 70 |

(b) Pricing model configuration

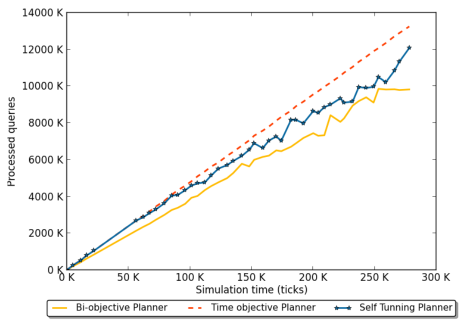
Table 2: Simulation environment configuration

veloped using Python and SimPy framework. All presented experiments are derived from simulation. There where four devices defined in environment: CPU1, CPU2, GPU1, GPU2. Data transfer bandwidth between CPU* $\leftrightarrow$ GPU* was measured on real system, bandwidth of GPU1 $\leftrightarrow$ GPU2 was calculated using CPU1 as transit device. Following weights in bi-objective scalarization were used $w_t = w_c = 1$ (i.e., without any preferences setting). Other settings of the simulation environment are gathered in Tables 2b and 2a. Simulation environment generates new query sequences, when spawn event occurs. Spawn event is generated randomly in a fixed interval and generates randomly set of query sequences (with fixed maximum). Every query sequence consists of maximally six operations and operates on random data volume. In the simulation the processed data size has a direct (linear) influence on processing speed. Each device has got a limited number of resources; a database operation can be performed only if needed resources are available. In other cases the operation is waiting. After generating desired number of query sequences the simulation stops spawning of new tasks and waits until all generated query sequences are processed.

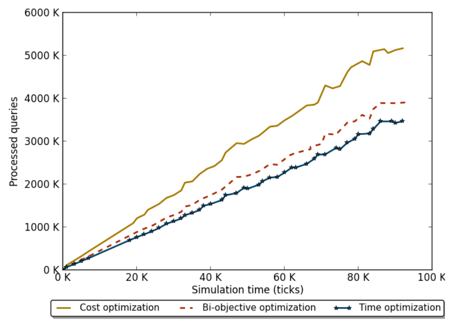
### 4.2 Simulation Results

Figure 1a presents simulated execution time of three scheduling frameworks processing a pool of generated sequences of queries. To each generated query sequence an optimization criterion (time, cost or bi-optimization) was assigned with equal probability 1/3. Optimization criteria are only used if query scheduling framework supports it, otherwise default criteria is used. All scheduling frameworks process exactly the same pool of generated query sequences.

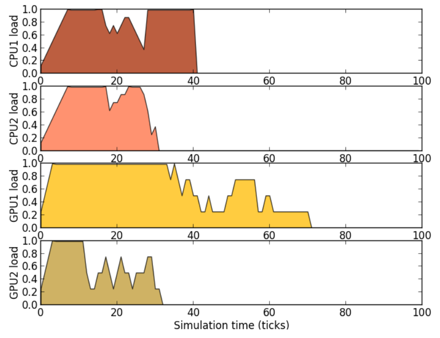
Compared frameworks are based on *OptimalSeq* algorithm but use different objective function. *Time objective planner* uses only $f_t$ and $g_t$ functions as optimization criteria; this means that it has no idea on load of each of devices. *Self Tuning Planner* is based on idea presented in Breß et. al. [3], i.e. it maintains a list of observed execution times on data $D$ for each algorithm $A^{o_i}_{u_j}$. Observations are interpolated (using e.q. cubic splines) to form new estimated execution time function. And finally *Bi-objective planner* is a proof of concept implementation of the model described in this work.



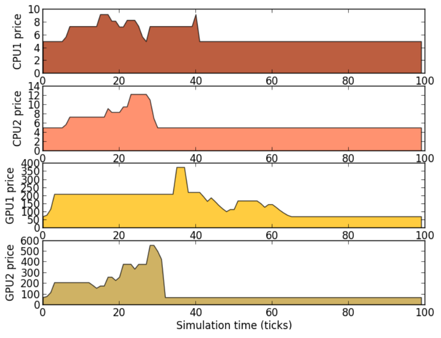
(a) Simulated efficiency of Bi-objective Heterogeneous Query Planer, Time based Query Planer and Self-Tuning Query Planner



(b) Efficiency of Bi-objective Heterogeneous Query Planer for various optimization tasks



(c) Simulated load of devices ($0 < load < 1$)



(d) Pricing of device

Fig. 1: Simulation results. Note that time is in simulation ticks.

As expected *Time Objective Planner* is the slowest one since it has no knowledge on the current load of devices. A solution suggested in [3] performs better. However, there are two problems with this approach: first it adds an additional overhead due to the interpolation of the observed execution times [3]; Secondly as may be observed in 1a it takes some time before it adapts to a new situation (in early stage it performs similarly to the *Time Objective Planner*). This is due the fact that it does not immediately react to load change of the device. Instead, it has to gather enough observations before adapting. The best performance is gained when using *Bi-objective planner*, this is due to the three types of optimization and the cost model which assures proper load balancing.

As our framework support different types of optimization in the Figure 1b, we present an impact of optimization type on processing performance. As it may be observed, time optimization is the most appropriate for query processing with high priority or with execution time constraint (like interactive queries or ad hoc data mining). Cost optimization is appropriate for operations with low priority or without time constraint (like batch processing or periodic jobs). Optimization of both cost and time (without preferences) leads to moderate processing speed but with better load balancing which is discussed later.

As proposed economic model is an important part of presented framework, in Figure 1 interaction between device load 1c and established device pricing 1d is illustrated. Notice how increased load influences unit pricing according to the formula 3. It is worth noting that pricing model may be tuned for specific applications (see Table 2b for this simulation settings).

### 4.3 Discussion

In Section 2.1 we cite three challenges of Hybrid Query Processing initially presented in Breß et.al. [3]. As our bi-objective optimization framework was designed in order to address this challenges, an evaluation in the context of the former mentioned problems is needed. We address *Critical Query* problem by allowing different optimization targets for queries. Choosing time optimisation allows to a priori articulate importance of a query. Also, the bi-objective optimization tends to promote queries which may gain more on particular devices (due to the cost-delay trade-off and the fact that the cost objective is designed to promote tasks with greater speed-up 3.3). The problem of *Execution Time Prediction* is addressed indirectly with bi-objective optimisation. This is because the bi-objective optimisation combines the cost objective function, which uses a current device load when pricing a device, with the execution time objective. So in most cases it is preferred to optimize both cost and time (without preferences towards any) through time/cost trade-off. Lastly different types of optimization apply also to *Optimization Impact* challenge. Choosing optimization criteria specifies a possible impact on other queries. Although, the preliminary results are promising and seem to confirm this, an extended evaluation is needed in future.

## 5 Related Work

Multiobjective query optimization was considered i.a. in Stonebraker et.al. [17] where a wide-area distributed database system (called Mariposa) was presented. An economic auction model was used as cost model. To process a query a user supplied a cost-delay trade-off curve. Because defining this kind of input data was problematic Papadimitriou et.al. proposed a new approach where an algorithm for finding $\epsilon$-Pareto optimal solutions was presented. The solution was that a user would manually choose one of presented solutions. This work differs both in an optimisation method and an economic model involved.

In our framework a user supplies an optimization objective for a query a priori (time, cost or bi-objective). Also as our model addresses the optimisation of co-processing interaction a simpler commodity market model could be used instead of a bidding model.

An extended overview on utilization of a GPU as a coprocessor in database operations may be found in [3]. Breß et. al. [3] proposed a framework for optimisation of hybrid CPU/GPU query plans and present two algorithms for constructing hybrid query sequences. The first algorithm selected the fastest algorithm for every element of a query sequence (including the cost of transfer between devices) with complexity $O(n)$. Unfortunately, this algorithm had two flaws [3]: the constructed plan could generate too frequent data transfers between devices, which may significantly affect the performance of data processing and also an optimal plan was not always generated. To overcome those problems they proposed the second algorithm. It searched for a continuous segment of operations on GPU that could improve the base CPU sequence. In order to find an optimal solution this algorithm generates all possible GPU sequences. Its complexity is obviously higher: $O(n^2)$. Our work extends this approach by allowing possible many various co-processing devices (Heterogeneous Query Planer in Section 3.1). Secondly our work incorporates commodity market model as well as bi-objective optimisation for better performance overcoming problems mentioned in 2.1. Additionally, the algorithm $OptimalSeq$ presented in our work may be used to produce a similar solution as the second algorithm by Breß et.al.[3] but with better complexity (in case of two devices $O(n)$).

It is worth to mention two surveys: the first one describing economic models in grid computing [6] and the second one describing methods for multi-objective optimisation [11].

## 6 Conclusions and Future Work

In this paper, we proposed a bi-objective optimization framework for heterogeneous query plans. We also presented an algorithm for creating query sequences in a heterogeneous environment with a single objective. This algorithm may be used to construct query sequences similar to [3] but with better complexity. For the purposes of this bi-objective optimization we designed a model including time and cost objectives function. The cost objective function and pricing model is build on foundations of commodity market economic model.

The preliminary experiments are very promising. We achieved good load balancing of the simulated devices combined with better optimization results.

In future work, an extended evaluation of the presented framework is needed, including; examination of parameters' influence on the model behaviour, careful assessment against Hybrid Query challenges. Another interesting field is extension of this model beyond CPU/GPU co-processing. Finally, the framework will be evaluated in a prototype time-series database [14,13].

## References

1. J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, Sept. 1984.
2. S. Breß, F. Beier, H. Rauhe, E. Schallehn, K.-U. Sattler, and G. Saake. Automatic selection of processing units for coprocessing in databases. In *Advances in Databases and Information Systems*, pages 57–70. Springer, 2012.

3. S. Breß, I. Geist, E. Schallehn, M. Mory, and G. Saake. A framework for cost based optimization of hybrid cpu/gpu query plans in database systems. *Control and Cybernetics*, pages 27–35, 2013.
4. S. Breß, S. Mohammad, and E. Schallehn. Self-tuning distribution of db-operations on hybrid cpu/gpu platforms. *Grundlagen von Datenbanken, CEUR-WS*, pages 89–94, 2012.
5. S. Breß, E. Schallehn, and I. Geist. Towards optimization of hybrid cpu/gpu query plans in database systems. In *New Trends in Databases and Information Systems*, pages 27–35. Springer, 2013.
6. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542, 2002.
7. W. Fang, B. He, and Q. Luo. Database compression on graphics processors. *Proceedings of the VLDB Endowment*, 3(1-2):670–680, 2010.
8. D. Florescu and D. Kossmann. Rethinking cost and performance of database systems. *ACM Sigmod Record*, 38(1):43–48, 2009.
9. M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In *ACM SIGMOD Record*, volume 25, pages 149–160. ACM, 1996.
10. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, 2000.
11. R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.
12. C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 52–59. ACM, 2001.
13. P. Przymus and K. Kaczmarski. Dynamic compression strategy for time series database using gpu. In *New Trends in Databases and Information Systems.* 17th East-European Conference on Advances in Databases and Information Systems September 1-4, 2013 - Genoa, Italy, 2013.
14. P. Przymus and K. Kaczmarski. Time series queries processing with gpu support. In *New Trends in Databases and Information Systems.* 17th East-European Conference on Advances in Databases and Information Systems September 1-4, 2013 - Genoa, Italy, 2013.
15. A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, 2009.
16. O. O. Sonmez and A. Gursoy. Comparison of pricing policies for a computational grid market. In *Parallel Processing and Applied Mathematics*, pages 766–773. Springer, 2006.
17. M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):48–63, 1996.
18. T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The implementation and performance of compressed databases. *ACM SIGMOD Record*, 29(3):55–67, 2000.

# Application of Wavelets and Kernel Methods to Detection and Extraction of Behaviours of Freshwater Mussels*

Piotr Przymus[1], Krzysztof Rykaczewski[1], and Ryszard Wiśniewski[2]

[1] Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University, Toruń, Poland
[2] Laboratory of Applied Hydrobiology,
Nicolaus Copernicus University, Toruń, Poland

**Abstract.** Some species of mussels are well-known bioindicators and may be used to create a Biological Early Warning System. Such systems use long-term observations of mussels activity for monitoring purposes. Yet, many of these systems are based on statistical methods and do not use all the potential that stays behind the data derived from the observations. In the paper we propose an algorithm based on wavelets and kernel methods to detect behaviour events in the collected data. We present our algorithm together with a discussion on the influence of various parameters on the received results. The study describes obtaining and pre-processing raw data and a feature extraction algorithm. Other papers which applied mathematical apparatus to Biological Early Warning Systems used much simpler methods and their effectiveness was questionable. We verify the results using a system with prepared tags for specified events. This leads us to a classification of these events and creating a *Dreissena polymorpha* behaviour dictionary and a Biological Early Warning System. Results from preliminary experiments show, that such a formulation of the problem, allows extracting relevant information from a given signal and yields an effective solution of the considered problem.

**Keywords:** Automated biomonitoring, Biological Early Warning System, Wavelets, Time series, Zebra mussel (Dreissena polymorpha).

## 1 Introduction

Monitoring of water contamination is one of the most crucial aspects of environmental and health care. Many existing monitoring systems examine water only for a narrow range of substances and work without continuous control. For that reason, systems based on life organisms, i.e. *Biological Early Warning Systems* (BEWS), increasingly gain interest and popularity. Building BEWS is a complex

* This work was supported in part by the Marshall of Kuyavian-Pomeranian Voivodeship in Poland with the funds from European Social Fund (EFS) (a part of integrated operational program for regional development, activity 2.6) in the form of a grant for PhD students (Step in the future program, second edition).

task, which requires a choice of a relevant bioindicator for the monitored environment, preparation of an activity measuring system, that will provide data for further processing, developing analysis and characterisation methods.

As aquatic organisms are sensitive to the concentration changes of different life supporting substances or the presence of xenobiotic, stressing or toxic compounds in the water, they are eligible as bioindicators. Most frequently used as sensing elements are cladocerans [1, 2], amphipods [3], bivalves [4, 5, 6], aquatic insects (*Chironomidae*) larvae [7] and fish [8]. Especially mussels, like *Mytilus* or *Dreissena*, as sessile bivalves, are very suitable for long-term, *in situ* water quality monitoring.

There are several methods for measuring the response of mussels to stressing factors. In older systems it was measured as a frequency of shell closing-opening events, through gluing wires to both halves of the shell and connecting them through the interface to a computer [9, 10]. The number of closed mussels in a treated group, in comparison to control, was a measure of stress response. More recently, a wire was replaced by a magnetic coil (or Hall sensor), on one valve, and a magnet on the other [11]. The value of the amplified signal was proportional to the distance (gape) between the two valves. As a response to stress of a tested group, the average value of gape in comparison to control mussels was measured. Both systems have limitations in informative and interpretative value of generated data. Our observations of *Dreissena polymorpha* mussels behaviour showed, that the response to stressing factor is more complex. The sequence of elementary events, i.e. an extent of gape change value and time of the return to the initial gape value can form specific patterns for various natural or stress caused activity rhythms. The presence of such rhythms was confirmed in [12].

In this paper, we propose an algorithmic, fully automated analysis method for extraction of the behaviour of the zebra mussels (*Dreissena polymorpha*). Because the behaviour of the zebra mussels is recorded as long series of shell states, logged every second, we needed an efficient analytic tools [7, 13]. For this reason, we applied mathematical apparatus of wavelets and kernel methods.

Detection of signal changes using the methods for spectral decomposition of time series is especially interesting. Fourier Transform (FT) technique can be applied to analyse the frequency spectrum, but it does not provide any insight into when a frequency component is present. In other words, we gain no information about either the time at which peak occurs or its duration in time (i.e. *localization in time*). Because of the limitations of the FT technique, we recommend using the wavelet transforms for investigating the long-term records of sudden changes in animal behaviour. Moreover, this approach may be used to dissecting the impact of unexpected events such as disruption in electric circuits.

The paper is organised as follows. In Section 2 we focus on obtaining and pre-processing raw data. Section 3 is devoted to give necessary background of wavelet theory. In Section 4 we present our behaviour extraction algorithm, which effectiveness is evaluated in Section 5. Finally, in two last Sections 6 and 7, we point out used programming tools and conclude discussing the results.

## 2 Materials and Methods

**Biological Signal.** Signal is a record of activity of freshwater mussels. We measure the changes of the distance between the valves. For a sample signal acquired from zebra mussel see Figure 2 on page 50.

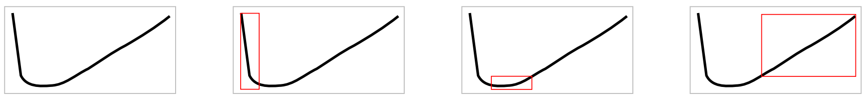
We want to extract single motions of mussels to classify them. It was proved in [12] that there are complex rhythms in the behaviour of *Dreissena polymorpha*. For example, the wanted pattern (shape of the graph of behaviour) may include the following phases: closing, resting and opening. These stages are presented in Figure 1 on page 46. Moreover, apart from the closing and opening phases, a vibration may occur. These are reactions to a stress or living activities. We search for time series fragments with following properties: at least closing and opening phases must appear, resting phase is optional; all phases may include perturbations. We analyse data with 16 minutes and 40 seconds (1000 seconds) periods of activity which from now on will be called *fragments*.

**Measuring System.** In our system, there are 8 mussels, which are located in a flow-through aquarium. They are attached to the ground, which does not affect their behaviour because of their sedentary nature.

We measure the changes of a magnetic field of a magnet placed on one side of the shell with a sensor placed on the other part of the shell. Data is collected every second from the sensors and transmitted to a database. The result sets showed, that the first prototype generates quite noisy data. Therefore, the measuring system should be improved in the future. The main difference between the old system and the new system will be based on used components type, their size and other resistance to interference from environment.

**Obtaining and Pre-processing Raw Data.** Denoising and data preparation steps consists of pre-processing filtering, removing white noise and averaging phase. A particularly important class of linear time-invariant systems are filters [14]. When the term frequency selective filter is used, it means that a system passes specific frequency components and totally rejects all others. This recommendation is common in particular for frequency selective filters like low-pass, band-pass and high-pass filters. A high-pass filter passes high frequencies well, but reduces the frequencies lower than the cut-off frequency. The real attenuation amount for each frequency differs from filter to filter. In our study, we used wavelet filter, which is high-pass filter and will be described later.

**Analysis Method.** Other papers investigated frequencies of closing-opening events [9, 10]. Previous results of observations conducted by the *Laboratory of Applied Hydrobiology at Nicolaus Copernicus University* reveal that one is able to extract motions as presented in Section 2 and, based on the activity record, is able to successfully assign water pollution to appropriate behaviour of mussel [9]. Stressful situation affects them, but it does not have to be a very violent reaction. For example, cyanotoxins and herbicides provide a recognisable, but not very intense, reaction. Therefore, we decided to analyse the normal activity



(a) Pattern (b) Opening (c) Resting (d) Closing

**Fig. 1.** Elementary phases of *Dreissena polymorpha* behaviour from time series point of view

and activity in a stressful situation to determine whether there is a difference and how it emerges.

One approach is to use statistical deviation. As abnormal we can take something that happens infrequently [8]. Not all incidents of behaviour, that do not conform to the norm, show an abnormality that we are looking for. We try to identify the *frame* (which will be defined in Section 4) in terms of shape, but not in terms of intensity or range of temporary phenomena.

In the future, we want to go even further and beyond the aforementioned situations to take an analysis of the captured events.

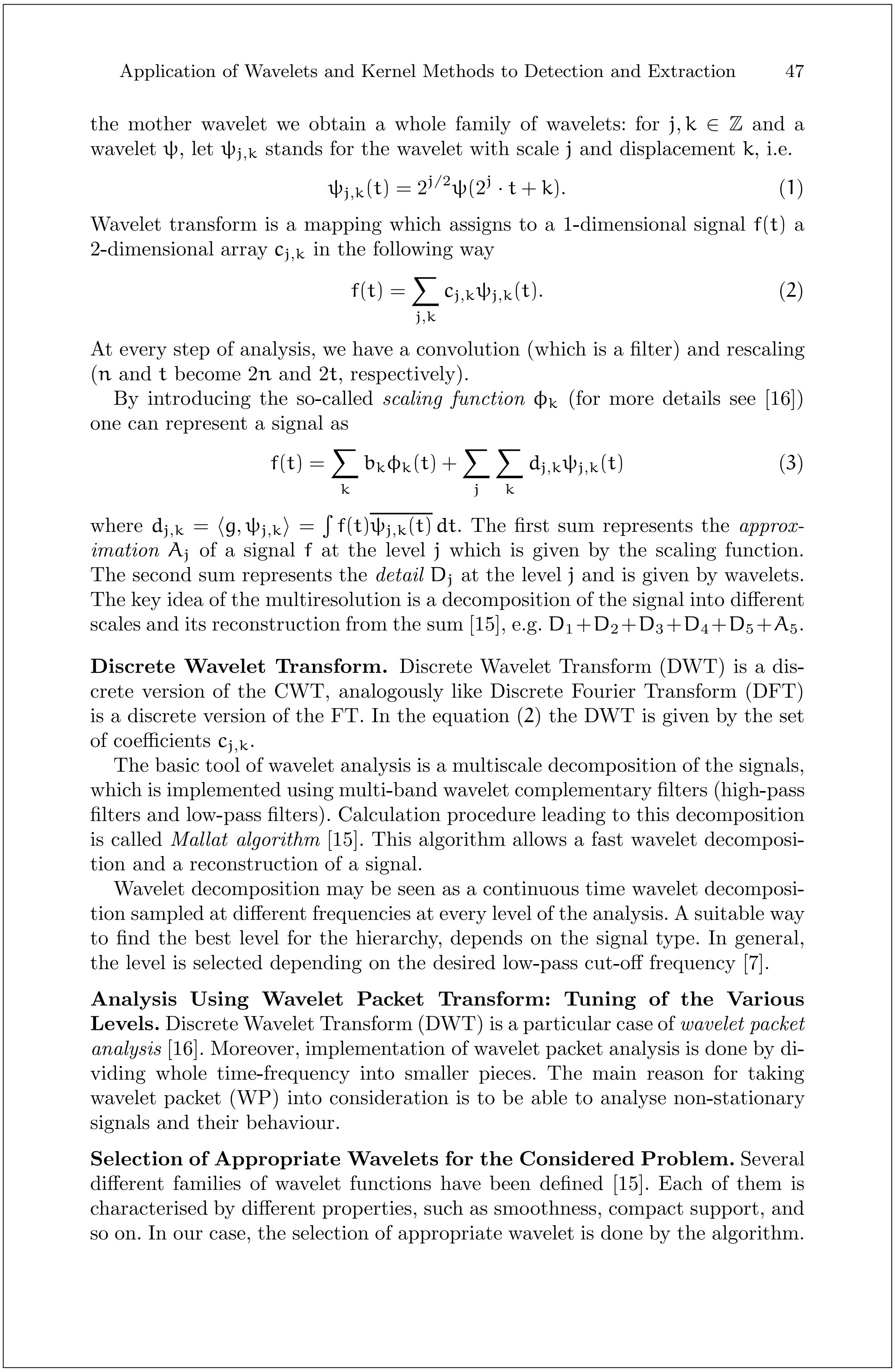
## 3 An Overview of the Wavelet Theory

The fundamental theory of wavelets was put forward by Haar in 1909 and then developed at the end of the 1960s. Now it has been extensively documented [15]. Wavelets have been very successful as an analytical tool to represent signals, in denoising, data compression and in time-scale analysis of time series, to mention a few of their applications. We refer inquisitive reader for more details concerning wavelet theory to [16].

Results obtained by this method are better than these obtained by Fourier analysis or other filter methods [15]. Because wavelet transforms can be exploited to analyse even non-stationary signals, they have been used in many medical applications and have been successful alternative methods to Fourier analysis.

Wavelets, in contrast to the Fourier Transform, are examples of *Multiscale Resolution Analysis*, which means that wavelet coefficients contain at once information about the frequency and the time domain. Thanks to this, they are particularly useful where the knowledge of these two characteristics of the signal is needed at the same time [16].

**Continuous Case.** Historically, wavelet analysis begins with *Continuous Wavelet* Transform (CWT). It provides a time-scale representation of a continuous function where the scale plays a role analogous to the one of the frequency in the analysis with the well-known Fourier Transform (FT). The main wavelet (the so-called *mother wavelet*) is a real valued function, that satisfies the following relations:$\int |\psi(t)|^2 dt = 1$ (quickly disappears), $\int \psi(t) dt = 0$ (oscillates). There are two main operations on wavelets: shift and rescaling. By applying them to



the mother wavelet we obtain a whole family of wavelets: for $j, k \in \mathbb{Z}$ and a wavelet $\psi$, let $\psi_{j,k}$ stands for the wavelet with scale $j$ and displacement $k$, i.e.

$$\psi_{j,k}(t) = 2^{j/2}\psi(2^j \cdot t + k). \tag{1}$$

Wavelet transform is a mapping which assigns to a 1-dimensional signal $f(t)$ a 2-dimensional array $c_{j,k}$ in the following way

$$f(t) = \sum_{j,k} c_{j,k}\psi_{j,k}(t). \tag{2}$$

At every step of analysis, we have a convolution (which is a filter) and rescaling ($n$ and $t$ become $2n$ and $2t$, respectively).

By introducing the so-called *scaling function* $\phi_k$ (for more details see [16]) one can represent a signal as

$$f(t) = \sum_k b_k\phi_k(t) + \sum_j \sum_k d_{j,k}\psi_{j,k}(t) \tag{3}$$

where $d_{j,k} = \langle g, \psi_{j,k}\rangle = \int f(t)\overline{\psi_{j,k}(t)}\,dt$. The first sum represents the *approximation* $A_j$ of a signal $f$ at the level $j$ which is given by the scaling function. The second sum represents the *detail* $D_j$ at the level $j$ and is given by wavelets. The key idea of the multiresolution is a decomposition of the signal into different scales and its reconstruction from the sum [15], e.g. $D_1+D_2+D_3+D_4+D_5+A_5$.

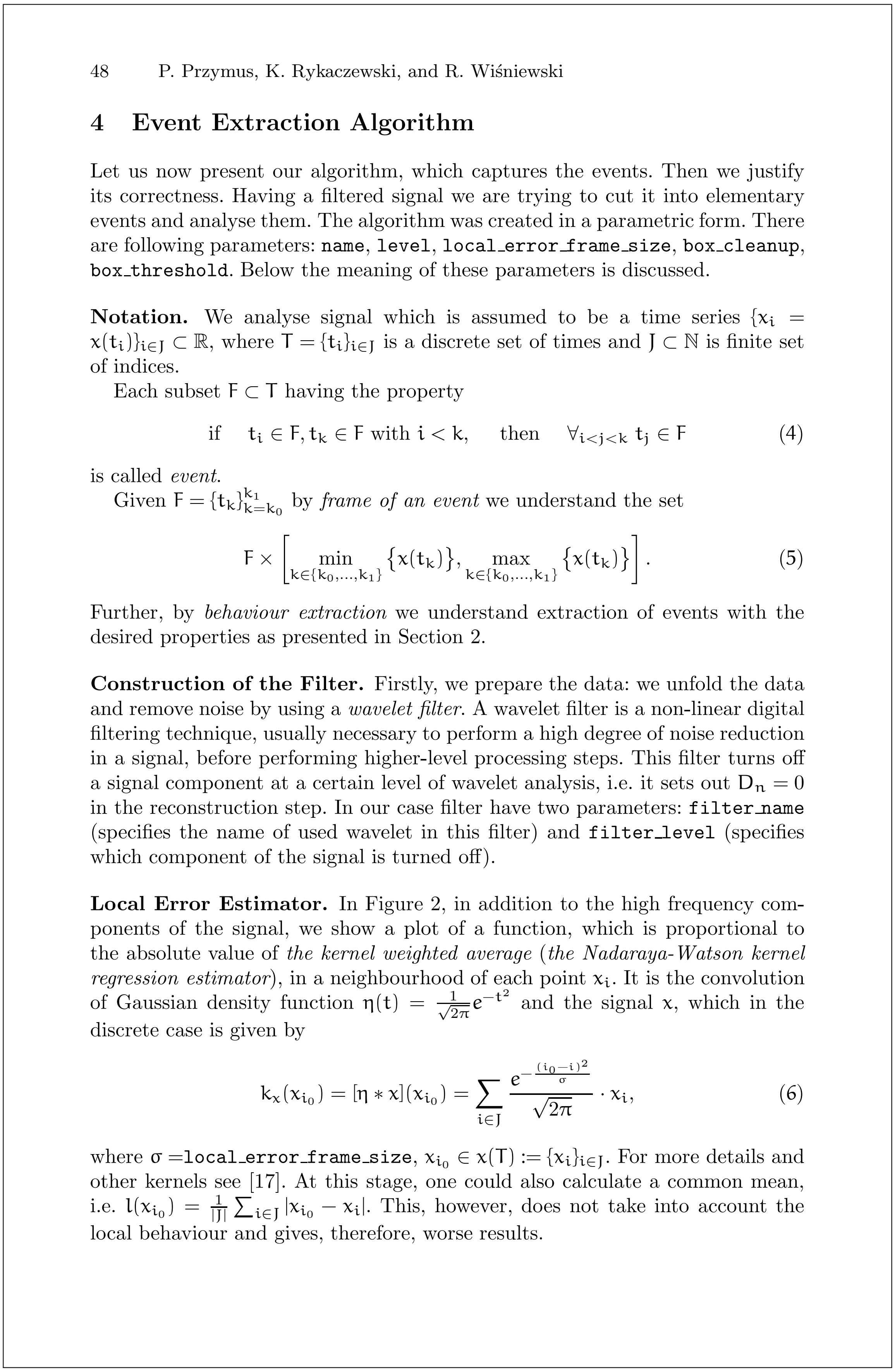
**Discrete Wavelet Transform.** Discrete Wavelet Transform (DWT) is a discrete version of the CWT, analogously like Discrete Fourier Transform (DFT) is a discrete version of the FT. In the equation (2) the DWT is given by the set of coefficients $c_{j,k}$.

The basic tool of wavelet analysis is a multiscale decomposition of the signals, which is implemented using multi-band wavelet complementary filters (high-pass filters and low-pass filters). Calculation procedure leading to this decomposition is called *Mallat algorithm* [15]. This algorithm allows a fast wavelet decomposition and a reconstruction of a signal.

Wavelet decomposition may be seen as a continuous time wavelet decomposition sampled at different frequencies at every level of the analysis. A suitable way to find the best level for the hierarchy, depends on the signal type. In general, the level is selected depending on the desired low-pass cut-off frequency [7].

**Analysis Using Wavelet Packet Transform: Tuning of the Various Levels.** Discrete Wavelet Transform (DWT) is a particular case of *wavelet packet analysis* [16]. Moreover, implementation of wavelet packet analysis is done by dividing whole time-frequency into smaller pieces. The main reason for taking wavelet packet (WP) into consideration is to be able to analyse non-stationary signals and their behaviour.

**Selection of Appropriate Wavelets for the Considered Problem.** Several different families of wavelet functions have been defined [15]. Each of them is characterised by different properties, such as smoothness, compact support, and so on. In our case, the selection of appropriate wavelet is done by the algorithm.



## 4 Event Extraction Algorithm

Let us now present our algorithm, which captures the events. Then we justify its correctness. Having a filtered signal we are trying to cut it into elementary events and analyse them. The algorithm was created in a parametric form. There are following parameters: `name`, `level`, `local_error_frame_size`, `box_cleanup`, `box_threshold`. Below the meaning of these parameters is discussed.

**Notation.** We analyse signal which is assumed to be a time series $\{x_i = x(t_i)\}_{i \in J} \subset \mathbb{R}$, where $T = \{t_i\}_{i \in J}$ is a discrete set of times and $J \subset \mathbb{N}$ is finite set of indices.

Each subset $F \subset T$ having the property

$$\text{if} \quad t_i \in F, t_k \in F \text{ with } i < k, \quad \text{then} \quad \forall_{i<j<k}\, t_j \in F \tag{4}$$

is called *event.*

Given $F = \{t_k\}_{k=k_0}^{k_1}$ by *frame of an event* we understand the set

$$F \times \left[ \min_{k \in \{k_0, \ldots, k_1\}} \{x(t_k)\}, \max_{k \in \{k_0, \ldots, k_1\}} \{x(t_k)\} \right]. \tag{5}$$

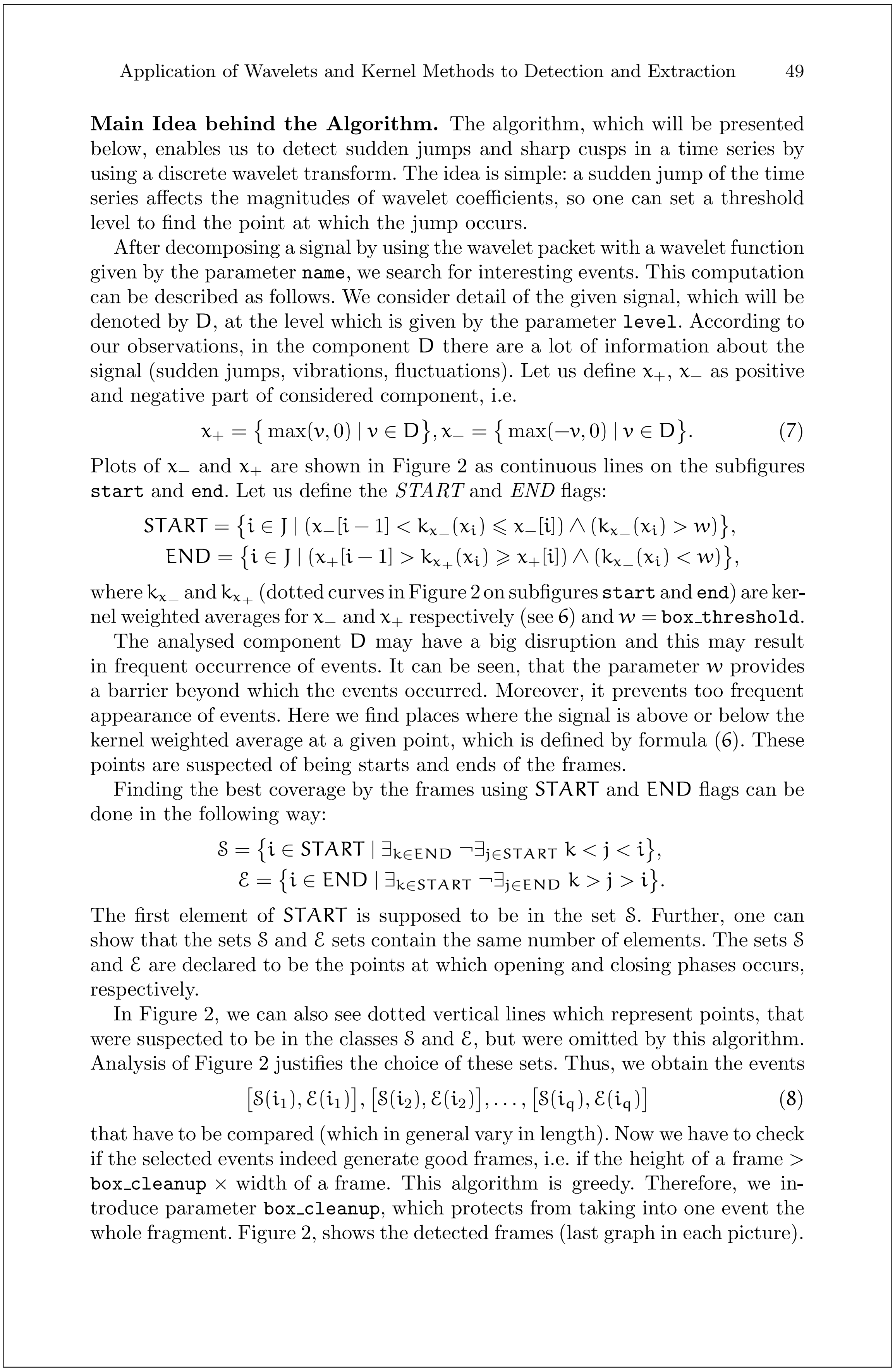
Further, by *behaviour extraction* we understand extraction of events with the desired properties as presented in Section 2.

**Construction of the Filter.** Firstly, we prepare the data: we unfold the data and remove noise by using a *wavelet filter.* A wavelet filter is a non-linear digital filtering technique, usually necessary to perform a high degree of noise reduction in a signal, before performing higher-level processing steps. This filter turns off a signal component at a certain level of wavelet analysis, i.e. it sets out $D_n = 0$ in the reconstruction step. In our case filter have two parameters: `filter_name` (specifies the name of used wavelet in this filter) and `filter_level` (specifies which component of the signal is turned off).

**Local Error Estimator.** In Figure 2, in addition to the high frequency components of the signal, we show a plot of a function, which is proportional to the absolute value of *the kernel weighted average* (*the Nadaraya-Watson kernel regression estimator*), in a neighbourhood of each point $x_i$. It is the convolution of Gaussian density function $\eta(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2}$ and the signal $x$, which in the discrete case is given by

$$k_x(x_{i_0}) = [\eta * x](x_{i_0}) = \sum_{i \in J} \frac{e^{-\frac{(i_0 - i)^2}{\sigma}}}{\sqrt{2\pi}} \cdot x_i, \tag{6}$$

where $\sigma =$`local_error_frame_size`, $x_{i_0} \in x(T) := \{x_i\}_{i \in J}$. For more details and other kernels see [17]. At this stage, one could also calculate a common mean, i.e. $l(x_{i_0}) = \frac{1}{|J|} \sum_{i \in J} |x_{i_0} - x_i|$. This, however, does not take into account the local behaviour and gives, therefore, worse results.



**Main Idea behind the Algorithm.** The algorithm, which will be presented below, enables us to detect sudden jumps and sharp cusps in a time series by using a discrete wavelet transform. The idea is simple: a sudden jump of the time series affects the magnitudes of wavelet coefficients, so one can set a threshold level to find the point at which the jump occurs.

After decomposing a signal by using the wavelet packet with a wavelet function given by the parameter `name`, we search for interesting events. This computation can be described as follows. We consider detail of the given signal, which will be denoted by $D$, at the level which is given by the parameter `level`. According to our observations, in the component $D$ there are a lot of information about the signal (sudden jumps, vibrations, fluctuations). Let us define $x_+$, $x_-$ as positive and negative part of considered component, i.e.

$$x_+ = \big\{ \max(v, 0) \mid v \in D \big\}, x_- = \big\{ \max(-v, 0) \mid v \in D \big\}. \tag{7}$$

Plots of $x_-$ and $x_+$ are shown in Figure 2 as continuous lines on the subfigures `start` and `end`. Let us define the *START* and *END* flags:

$$\begin{aligned} \mathsf{START} &= \big\{ i \in J \mid (x_-[i-1] < k_{x_-}(x_i) \leqslant x_-[i]) \wedge (k_{x_-}(x_i) > w) \big\}, \\ \mathsf{END} &= \big\{ i \in J \mid (x_+[i-1] > k_{x_+}(x_i) \geqslant x_+[i]) \wedge (k_{x_-}(x_i) < w) \big\}, \end{aligned}$$

where $k_{x_-}$ and $k_{x_+}$ (dotted curves in Figure 2 on subfigures `start` and `end`) are kernel weighted averages for $x_-$ and $x_+$ respectively (see 6) and $w =$ `box_threshold`.

The analysed component $D$ may have a big disruption and this may result in frequent occurrence of events. It can be seen, that the parameter $w$ provides a barrier beyond which the events occurred. Moreover, it prevents too frequent appearance of events. Here we find places where the signal is above or below the kernel weighted average at a given point, which is defined by formula (6). These points are suspected of being starts and ends of the frames.

Finding the best coverage by the frames using $\mathsf{START}$ and $\mathsf{END}$ flags can be done in the following way:

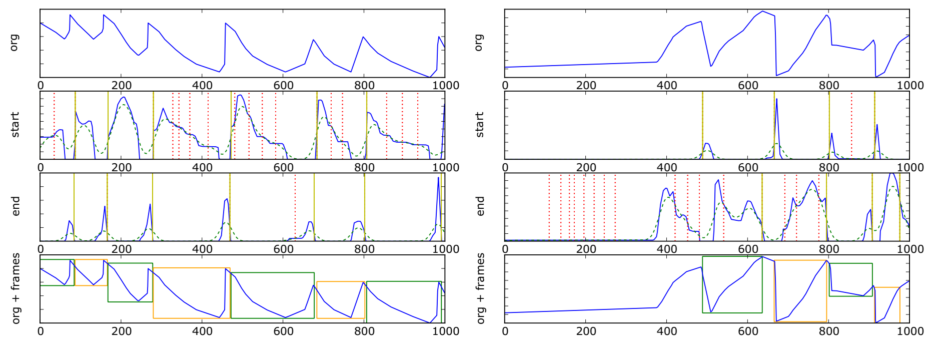
$$\begin{aligned} \mathcal{S} &= \big\{ i \in \mathsf{START} \mid \exists_{k \in \mathsf{END}} \neg \exists_{j \in \mathsf{START}}\ k < j < i \big\}, \\ \mathcal{E} &= \big\{ i \in \mathsf{END} \mid \exists_{k \in \mathsf{START}} \neg \exists_{j \in \mathsf{END}}\ k > j > i \big\}. \end{aligned}$$

The first element of $\mathsf{START}$ is supposed to be in the set $\mathcal{S}$. Further, one can show that the sets $\mathcal{S}$ and $\mathcal{E}$ sets contain the same number of elements. The sets $\mathcal{S}$ and $\mathcal{E}$ are declared to be the points at which opening and closing phases occurs, respectively.

In Figure 2, we can also see dotted vertical lines which represent points, that were suspected to be in the classes $\mathcal{S}$ and $\mathcal{E}$, but were omitted by this algorithm. Analysis of Figure 2 justifies the choice of these sets. Thus, we obtain the events

$$\big[\mathcal{S}(i_1), \mathcal{E}(i_1)\big], \big[\mathcal{S}(i_2), \mathcal{E}(i_2)\big], \ldots, \big[\mathcal{S}(i_q), \mathcal{E}(i_q)\big] \tag{8}$$

that have to be compared (which in general vary in length). Now we have to check if the selected events indeed generate good frames, i.e. if the height of a frame $>$ `box_cleanup` $\times$ width of a frame. This algorithm is greedy. Therefore, we introduce parameter `box_cleanup`, which protects from taking into one event the whole fragment. Figure 2, shows the detected frames (last graph in each picture).



(a) (b)

**Fig. 2.** Frames extracted using our algorithm

## 5 Experiment Results

In this section, we will evaluate the effectiveness of the proposed algorithm, against the real data. All data were derived from experiments, in which impact of salt, herbicide and yeast on mussels was tested.

**Data Specification.** We used the data from 8 sensors with frequency of reading being 1 second. Each sensor recorded 11245 minutes of data which consist of reals from the interval $[0, 45]$. Sensor read error is assumed to be in the interval $[0, 3]$. Due to technical requirements the data set was split into time windows (fragments) of length 1000 seconds.

It is important to note that this data set is a subset from a bigger set of experiments, i.e. besides limiting data set, we only choose time fragments where height of the frame $>$ sensor read error.

Thus, in what follows, it is assumed, that everything what is below sensor read error is only a negligible perturbation. Considering this, we suggest using a more accurate sensor system. Some early experiments with better sensors (Hall sensors) show that our algorithm gains effectiveness.

The mussels were derived from the same colony. All biological experiments were conducted with at least 50% of mussels in control terms. There were also biological experiments conducted only in control conditions. Table 1 describes data set, obtained from this experiments.

**Result Evaluation.** The data set was independently analysed by two researchers who put markers in the areas of events described above. For comparing similarity between researchers and the algorithm we use *Tversky index* defined as follows:

$$S(X, Y) = \frac{\sum_{i,j} |X_i \cap Y_j|}{\sum_{i,j}(|X_i \cap Y_j| + \alpha|X_i - Y_j| + \beta|Y_i - X_j|)},$$

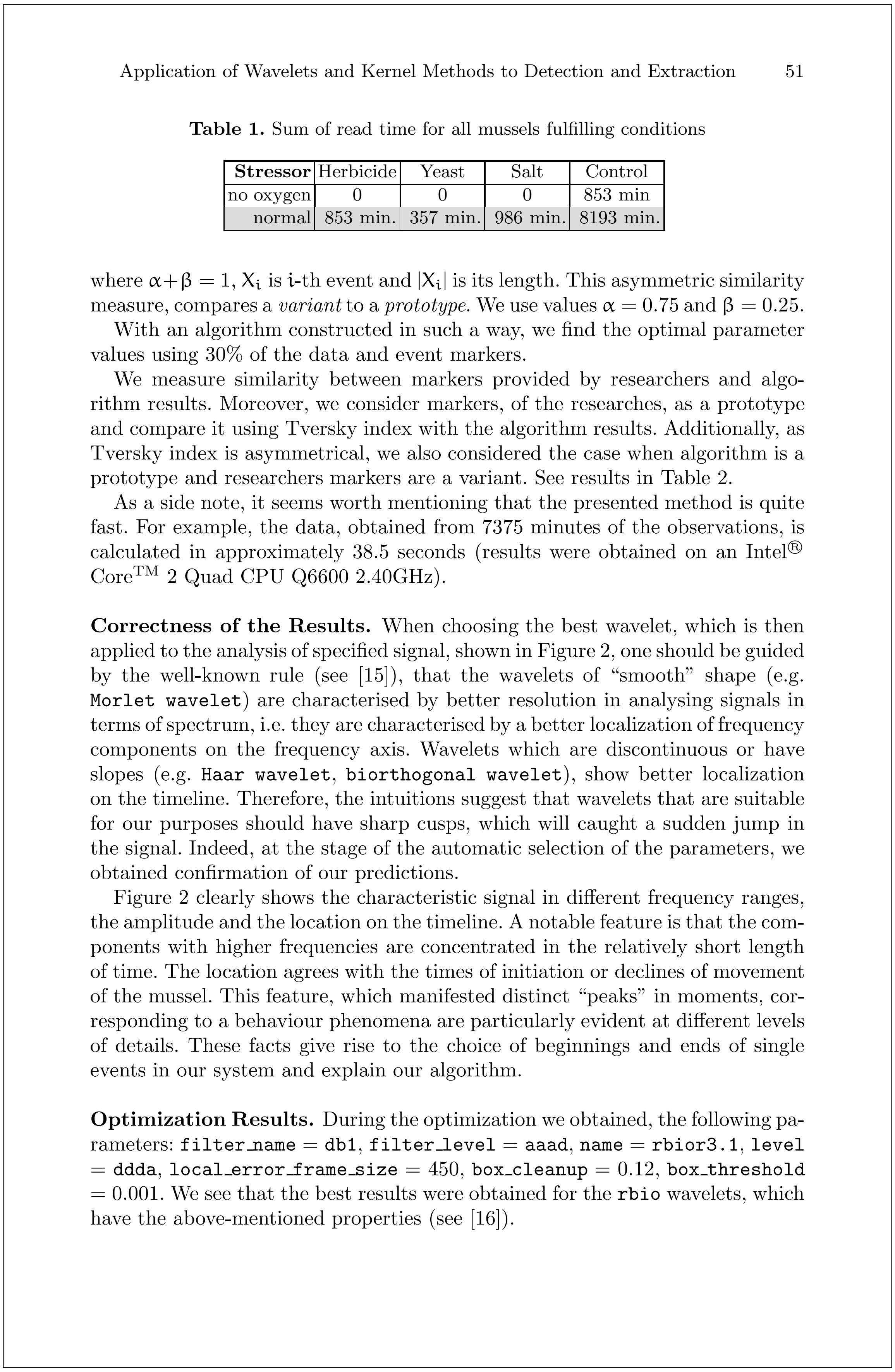


**Table 1.** Sum of read time for all mussels fulfilling conditions

| **Stressor** | Herbicide | Yeast | Salt | Control |
|---|---|---|---|---|
| no oxygen | 0 | 0 | 0 | 853 min |
| normal | 853 min. | 357 min. | 986 min. | 8193 min. |

where $\alpha+\beta = 1$, $X_i$ is $i$-th event and $|X_i|$ is its length. This asymmetric similarity measure, compares a *variant* to a *prototype*. We use values $\alpha = 0.75$ and $\beta = 0.25$.

With an algorithm constructed in such a way, we find the optimal parameter values using 30% of the data and event markers.

We measure similarity between markers provided by researchers and algorithm results. Moreover, we consider markers, of the researches, as a prototype and compare it using Tversky index with the algorithm results. Additionally, as Tversky index is asymmetrical, we also considered the case when algorithm is a prototype and researchers markers are a variant. See results in Table 2.

As a side note, it seems worth mentioning that the presented method is quite fast. For example, the data, obtained from 7375 minutes of the observations, is calculated in approximately 38.5 seconds (results were obtained on an Intel$^{®}$ Core$^{TM}$ 2 Quad CPU Q6600 2.40GHz).

**Correctness of the Results.** When choosing the best wavelet, which is then applied to the analysis of specified signal, shown in Figure 2, one should be guided by the well-known rule (see [15]), that the wavelets of "smooth" shape (e.g. `Morlet wavelet`) are characterised by better resolution in analysing signals in terms of spectrum, i.e. they are characterised by a better localization of frequency components on the frequency axis. Wavelets which are discontinuous or have slopes (e.g. `Haar wavelet`, `biorthogonal wavelet`), show better localization on the timeline. Therefore, the intuitions suggest that wavelets that are suitable for our purposes should have sharp cusps, which will caught a sudden jump in the signal. Indeed, at the stage of the automatic selection of the parameters, we obtained confirmation of our predictions.

Figure 2 clearly shows the characteristic signal in different frequency ranges, the amplitude and the location on the timeline. A notable feature is that the components with higher frequencies are concentrated in the relatively short length of time. The location agrees with the times of initiation or declines of movement of the mussel. This feature, which manifested distinct "peaks" in moments, corresponding to a behaviour phenomena are particularly evident at different levels of details. These facts give rise to the choice of beginnings and ends of single events in our system and explain our algorithm.

**Optimization Results.** During the optimization we obtained, the following parameters: `filter_name` = `db1`, `filter_level` = `aaad`, `name` = `rbior3.1`, `level` = `ddda`, `local_error_frame_size` = 450, `box_cleanup` = 0.12, `box_threshold` = 0.001. We see that the best results were obtained for the `rbio` wavelets, which have the above-mentioned properties (see [16]).

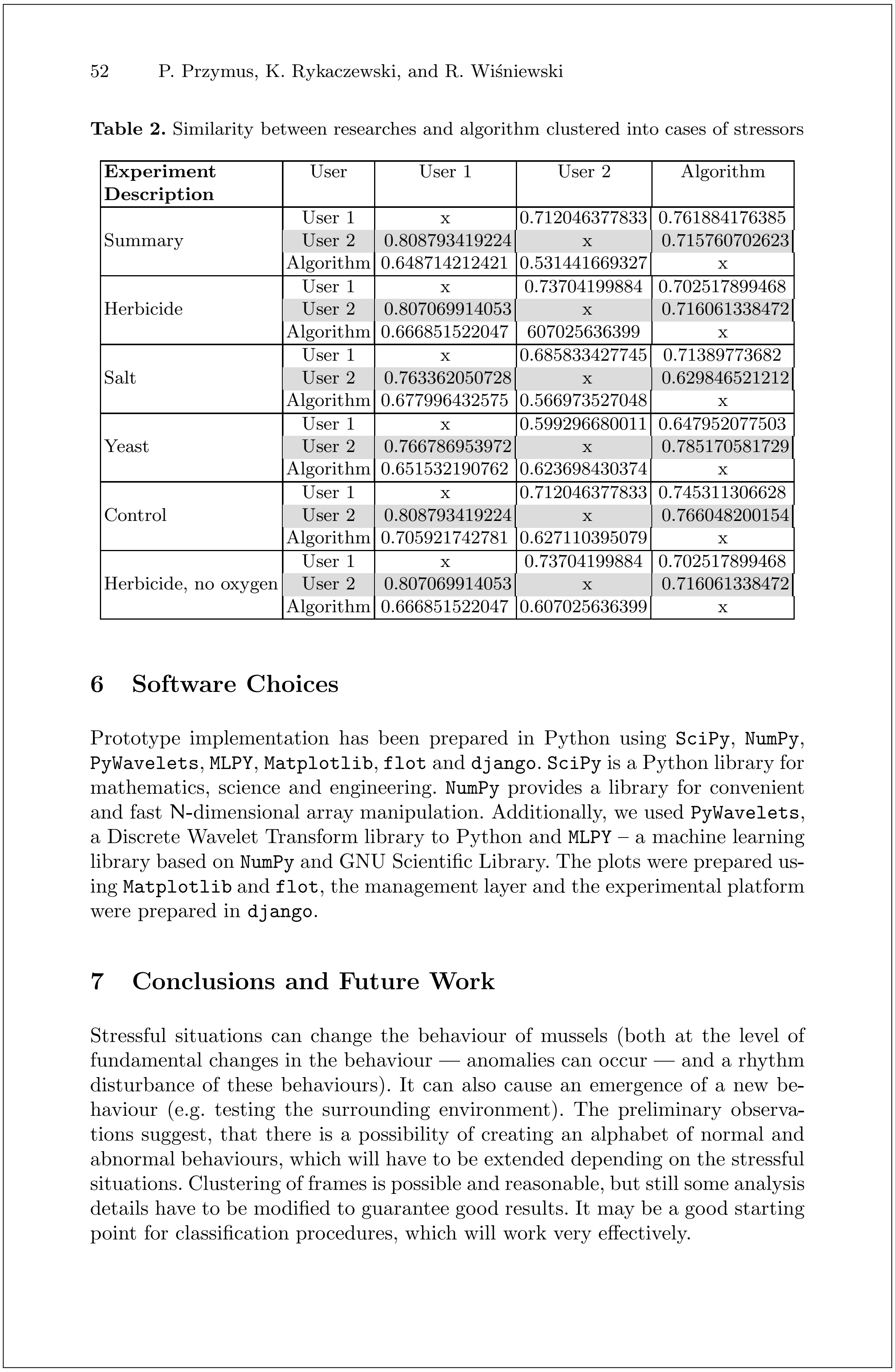


**Table 2.** Similarity between researches and algorithm clustered into cases of stressors

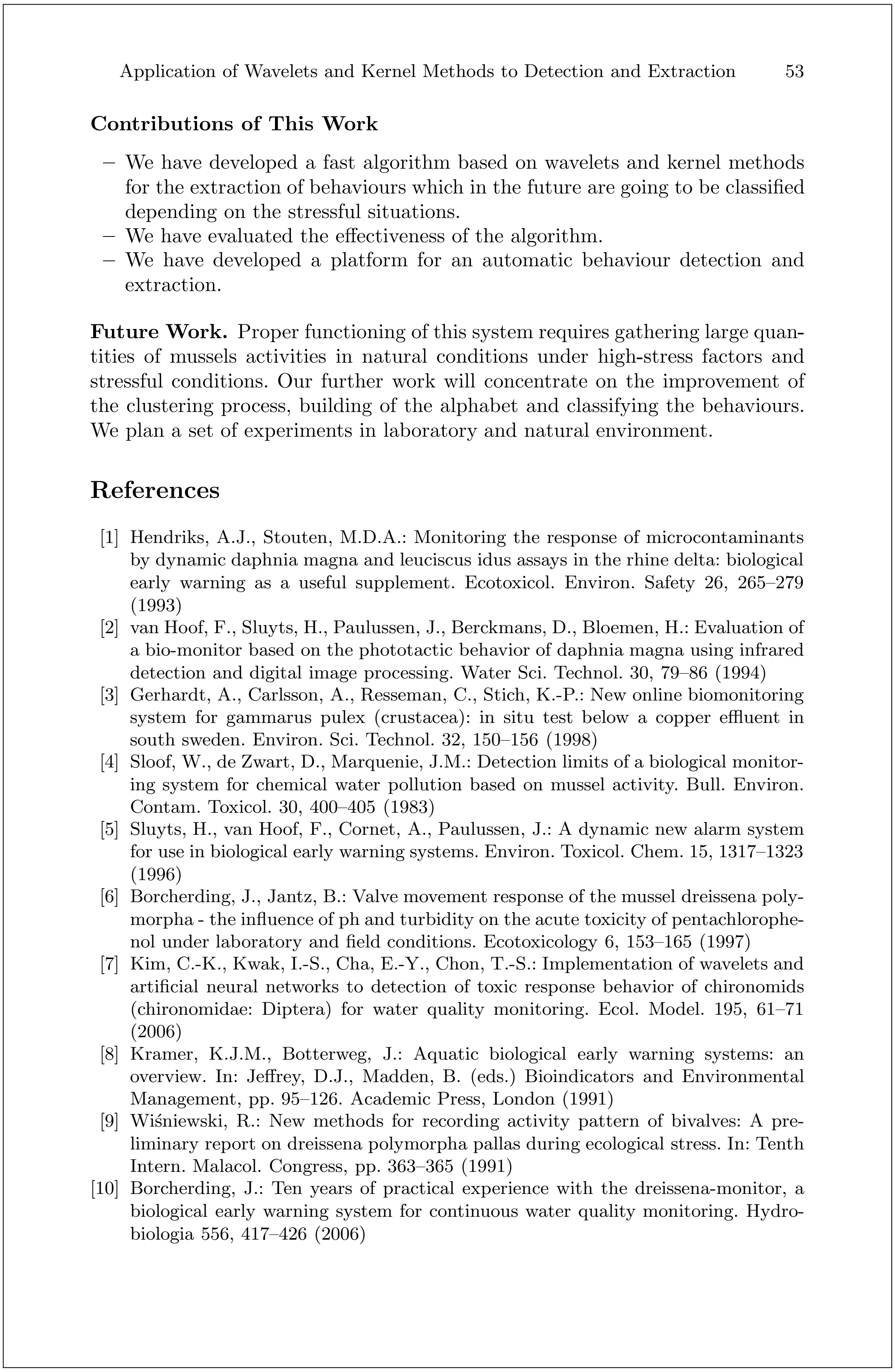
| **Experiment Description** | User | User 1 | User 2 | Algorithm |
|---|---|---|---|---|
| Summary | User 1 | x | 0.712046377833 | 0.761884176385 |
| | User 2 | 0.808793419224 | x | 0.715760702623 |
| | Algorithm | 0.648714212421 | 0.531441669327 | x |
| Herbicide | User 1 | x | 0.73704199884 | 0.702517899468 |
| | User 2 | 0.807069914053 | x | 0.716061338472 |
| | Algorithm | 0.666851522047 | 607025636399 | x |
| Salt | User 1 | x | 0.685833427745 | 0.71389773682 |
| | User 2 | 0.763362050728 | x | 0.629846521212 |
| | Algorithm | 0.677996432575 | 0.566973527048 | x |
| Yeast | User 1 | x | 0.599296680011 | 0.647952077503 |
| | User 2 | 0.766786953972 | x | 0.785170581729 |
| | Algorithm | 0.651532190762 | 0.623698430374 | x |
| Control | User 1 | x | 0.712046377833 | 0.745311306628 |
| | User 2 | 0.808793419224 | x | 0.766048200154 |
| | Algorithm | 0.705921742781 | 0.627110395079 | x |
| Herbicide, no oxygen | User 1 | x | 0.73704199884 | 0.702517899468 |
| | User 2 | 0.807069914053 | x | 0.716061338472 |
| | Algorithm | 0.666851522047 | 0.607025636399 | x |

## 6 Software Choices

Prototype implementation has been prepared in Python using `SciPy`, `NumPy`, `PyWavelets`, `MLPY`, `Matplotlib`, `flot` and `django`. `SciPy` is a Python library for mathematics, science and engineering. `NumPy` provides a library for convenient and fast N-dimensional array manipulation. Additionally, we used `PyWavelets`, a Discrete Wavelet Transform library to Python and `MLPY` – a machine learning library based on `NumPy` and GNU Scientific Library. The plots were prepared using `Matplotlib` and `flot`, the management layer and the experimental platform were prepared in `django`.

## 7 Conclusions and Future Work

Stressful situations can change the behaviour of mussels (both at the level of fundamental changes in the behaviour — anomalies can occur — and a rhythm disturbance of these behaviours). It can also cause an emergence of a new behaviour (e.g. testing the surrounding environment). The preliminary observations suggest, that there is a possibility of creating an alphabet of normal and abnormal behaviours, which will have to be extended depending on the stressful situations. Clustering of frames is possible and reasonable, but still some analysis details have to be modified to guarantee good results. It may be a good starting point for classification procedures, which will work very effectively.



**Contributions of This Work**

– We have developed a fast algorithm based on wavelets and kernel methods for the extraction of behaviours which in the future are going to be classified depending on the stressful situations.
– We have evaluated the effectiveness of the algorithm.
– We have developed a platform for an automatic behaviour detection and extraction.

**Future Work.** Proper functioning of this system requires gathering large quantities of mussels activities in natural conditions under high-stress factors and stressful conditions. Our further work will concentrate on the improvement of the clustering process, building of the alphabet and classifying the behaviours. We plan a set of experiments in laboratory and natural environment.

## References

[1] Hendriks, A.J., Stouten, M.D.A.: Monitoring the response of microcontaminants by dynamic daphnia magna and leuciscus idus assays in the rhine delta: biological early warning as a useful supplement. Ecotoxicol. Environ. Safety 26, 265–279 (1993)

[2] van Hoof, F., Sluyts, H., Paulussen, J., Berckmans, D., Bloemen, H.: Evaluation of a bio-monitor based on the phototactic behavior of daphnia magna using infrared detection and digital image processing. Water Sci. Technol. 30, 79–86 (1994)

[3] Gerhardt, A., Carlsson, A., Resseman, C., Stich, K.-P.: New online biomonitoring system for gammarus pulex (crustacea): in situ test below a copper effluent in south sweden. Environ. Sci. Technol. 32, 150–156 (1998)

[4] Sloof, W., de Zwart, D., Marquenie, J.M.: Detection limits of a biological monitoring system for chemical water pollution based on mussel activity. Bull. Environ. Contam. Toxicol. 30, 400–405 (1983)

[5] Sluyts, H., van Hoof, F., Cornet, A., Paulussen, J.: A dynamic new alarm system for use in biological early warning systems. Environ. Toxicol. Chem. 15, 1317–1323 (1996)

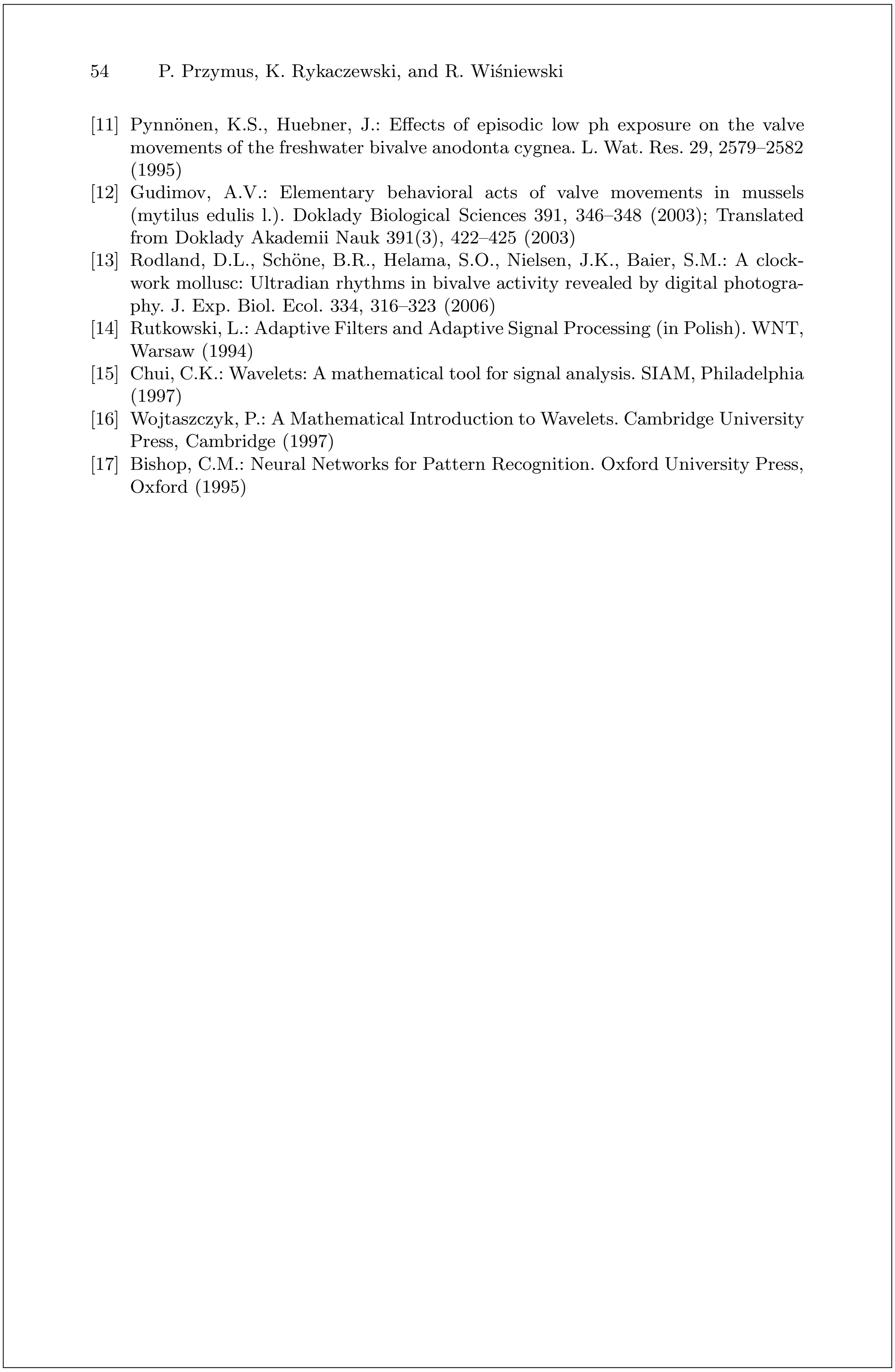
[6] Borcherding, J., Jantz, B.: Valve movement response of the mussel dreissena polymorpha - the influence of ph and turbidity on the acute toxicity of pentachlorophenol under laboratory and field conditions. Ecotoxicology 6, 153–165 (1997)

[7] Kim, C.-K., Kwak, I.-S., Cha, E.-Y., Chon, T.-S.: Implementation of wavelets and artificial neural networks to detection of toxic response behavior of chironomids (chironomidae: Diptera) for water quality monitoring. Ecol. Model. 195, 61–71 (2006)

[8] Kramer, K.J.M., Botterweg, J.: Aquatic biological early warning systems: an overview. In: Jeffrey, D.J., Madden, B. (eds.) Bioindicators and Environmental Management, pp. 95–126. Academic Press, London (1991)

[9] Wiśniewski, R.: New methods for recording activity pattern of bivalves: A preliminary report on dreissena polymorpha pallas during ecological stress. In: Tenth Intern. Malacol. Congress, pp. 363–365 (1991)

[10] Borcherding, J.: Ten years of practical experience with the dreissena-monitor, a biological early warning system for continuous water quality monitoring. Hydrobiologia 556, 417–426 (2006)



[11] Pynnönen, K.S., Huebner, J.: Effects of episodic low ph exposure on the valve movements of the freshwater bivalve anodonta cygnea. L. Wat. Res. 29, 2579–2582 (1995)
[12] Gudimov, A.V.: Elementary behavioral acts of valve movements in mussels (mytilus edulis l.). Doklady Biological Sciences 391, 346–348 (2003); Translated from Doklady Akademii Nauk 391(3), 422–425 (2003)
[13] Rodland, D.L., Schöne, B.R., Helama, S.O., Nielsen, J.K., Baier, S.M.: A clockwork mollusc: Ultradian rhythms in bivalve activity revealed by digital photography. J. Exp. Biol. Ecol. 334, 316–323 (2006)
[14] Rutkowski, L.: Adaptive Filters and Adaptive Signal Processing (in Polish). WNT, Warsaw (1994)
[15] Chui, C.K.: Wavelets: A mathematical tool for signal analysis. SIAM, Philadelphia (1997)
[16] Wojtaszczyk, P.: A Mathematical Introduction to Wavelets. Cambridge University Press, Cambridge (1997)
[17] Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)

Future Generation Computer Systems 33 (2014) 81–89

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Zebra mussels' behaviour detection, extraction and classification using wavelets and kernel methods

Piotr Przymus [a,*], Krzysztof Rykaczewski [a], Ryszard Wiśniewski [b]

[a] *Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Chopina 12/18, 87-100 Toruń, Poland*
[b] *Laboratory of Applied Hydrobiology, Nicolaus Copernicus University, Gagarina 9, 87-100 Toruń, Poland*

## HIGHLIGHTS

- We have developed a fully automated method for extraction and analysis of the behaviour of Dreissena polymorpha.
- We have evaluated usefulness of the feature set used for classification.
- We have proposed a framework for the classification of control and stress conditions for the purpose of the risk analysis.

ARTICLE INFO

*Article history:*
Received 15 June 2012
Received in revised form
11 April 2013
Accepted 13 April 2013
Available online 22 April 2013

*Keywords:*
Wavelet
Zebra mussel
Feature extraction
Classification
BEWS

ABSTRACT

This paper concerns the detection, feature extraction and classification of behaviours of *Dreissena polymorpha*. A new algorithm based on wavelets and kernel methods that detects relevant events in the collected data is presented. This algorithm allows us to extract elementary events from the behaviour of a living organism. Moreover, we propose an efficient framework for automatic classification to separate the control and stressful conditions.

## 1. Introduction

One of the most important problems connected with health and environmental protection is the monitoring of water pollution. However, many existing systems do not offer continuous monitoring and examine only a narrow range of substances. Therefore, systems based on living organisms (i.e. Biological Early Warning Systems, BEWS) have become increasingly popular recently [1–3]. Building such a system is a complex task, which requires a selection of an appropriate bioindicator for the monitored environment, preparation of a measurement system that provides data for further processing, analysis and classification methods.

It is well-known that aquatic organisms are sensitive to changes in concentrations of various substances (such as *xenobiotics* and other toxic compounds) in water, therefore can be successfully used as bioindicators. Among the most frequently used are: *cladocerans* [4,5], *amphipods* [6], *mussels* [7–9], the larvae of aquatic insects *(Chironomidae)* [10] and fish [11]. Especially mussels, like *Mytilus* or *Dreissena*, as sessile bivalves, are very suitable for long-term, *in situ* water quality monitoring.

There are several methods for measuring the response of mussels to stress factors. One option is to measure the frequency of shell closing–opening states, through gluing wires to both halves of the shell and connecting them through an interface to a computer [12,13]. The number of closed mussels in a treated group, in comparison to control one, is a measure of stress response. Another option is to use a magnetic coil (or a Hall sensor) on one valve, and a magnet on the other [14,15]. The value of the amplified signal is proportional to the distance (gape) between the two valves. Various factors are taken into consideration as a response to stress of a tested group, such as the difference of the average value of the gape in comparison to control mussels [15], increased activity or increased time in which the shells are closed or are open [14].

These solutions, however, neglect more complex behaviour of mussels, such as changes in elementary movements. The sequence

[*] Corresponding author. Tel.: +48 56 611 3463; fax: +48 56 611 2987.
*E-mail addresses:* eror@mat.umk.pl, piotr.przymus@gmail.com (P. Przymus), mozgun@mat.umk.pl (K. Rykaczewski), wisniew@biol.uni.torun.pl (R. Wiśniewski).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.04.015

of elementary events, i.e. the extent of gape change and the time of the first return to the initial gape value can form specific patterns for various natural or stress caused activity rhythms [16,17]. In some cases, the appearance of different patterns in elementary behaviour is the only difference compared to the control group showing appearance of hazardous substances [18].

Therefore, a fully automated method of analysis for the extraction of the behaviour of *Dreissena polymorpha* mussels is proposed and a framework (see Fig. 1) for risk analysis and classification of control and stress conditions based on elementary behaviour is presented. We discuss here our extraction algorithm based on wavelets and kernel methods. Then we show how to build a collection of observations based on a combination of our event extraction algorithm and the method proposed in [19] for EEG signals as well. Moreover, an assessment of the adequacy of the set of features used for classification is made. We carry out risk analysis and classification experiments using different classifiers ($k$-NN, SRDA, FDA) and the sets of features. Finally, we present and discuss the results.

Based on our extraction algorithm, a set of tools for automatic extraction and analysis of elementary events is proposed. We investigate the usefulness of the proposed extraction method as a tool to support laboratory work, which is an important improvement since this work was mostly done manually before.

The paper is partly an extension of [20]. It is organised as follows. The motivation of this work is presented in Section 2. Section 3 presents the data collection used in this work. Section 4 presents the theory of wavelets necessary in Section 5, where behaviour extraction algorithm is described. The evaluation of the effectiveness of the behaviour extraction algorithm is given in Section 6. The framework for automatic classification of control and stress conditions is discussed in Section 7. Finally, Section 8, summarise the results of the paper.

## 2. Motivation

Biomonitoring based on the behaviour of mussels, and particularly of the species *Dreissena polymorpha*, has been repeatedly tested in the laboratory and in natural conditions, and its effectiveness has been confirmed in many studies [12,14,15,13]. Many of the biomonitoring systems use *Dreissena polymorpha* as a bioindicator, but each of them operates on different principles. The system described in [12,13] monitors the activity of mussels by checking whether they are open or closed. A measure indicating the level of risk is the percentage of closed mussels in comparison to the percentage of closed mussels in the control group. In systems based on [14,15], the distance between valves is measured using Hall sensor and a magnet. Risk analysis is based on the following factors: the average level of valve distance, mussels activity (number of opening and closing events), and the average time during which the mussels are closed/open.

However, the existing systems do not exploit the full potential of mussels as a bioindicator. Experiments were conducted [18,17] to show that a change in activity of mussels may be more subtle in response to various concentrations of substances such as *cyanotoxins, herbicides, salt* or *LPS (Lipopolysaccharide)*. These publications dealt with an analysis of the life rhythms described in [16] and took under consideration changes that may appear in the elementary behaviour to suggest the appearance of harmful substances. A typical *elementary event* (or *behaviour*) consists of the following stages: closing, opening and resting (see Fig. 2, all these stages usually include some perturbations).

It turns out that the behaviour of the bioindicator could be slightly changed at the level of elementary behaviours. Moreover, it may be the only change compared to the control mussels. In practice, this means that the elementary event is disturbed in some of the three phases. Some examples of distorted elementary movements are placed in Fig. 3.

Our contribution in this area is the development of feature extraction algorithm, so that we can automatically extract elementary events from the behaviour of the mussels as described above. Then we assign to such behaviour a set of features used later in risk analysis.

Since the behaviour of the zebra mussels is recorded as a long series of shell states, logged at every second, we needed efficient analytic tools such as wavelets [10,21]. Note that Fourier Transform (FT) technique can be applied to analyse the frequency spectrum, but it does not provide any insight into which frequency component is present and when. Therefore, to fulfil the requirements of our system, we applied wavelets and kernel methods. These tools are better suited to study the long-term records of sudden changes in the animal behaviour; they can gain information about the time and duration of the peak in the reaction of the living being (see Section 5.4.1).

## 3. Data selection

In this section we describe a set of data originating from a biological experiment, which will be further analysed.

The data are a record of freshwater mussel activity, measured as the distance between the two halves of the shell. Mussels are in flow-through aquarium and are attached to its wall (due to the stationary lifestyle, it does not significantly affect their behaviour). A sensor measures changes in the magnetic field of the magnet placed on the one bivalve shell. The measurements are recorded with one second rate and are transmitted to the database. The system monitors eight mussels at the same time and each sensor generates a measurement from the interval $[0, 45]$ with a possible measurement error of $\pm 3$.

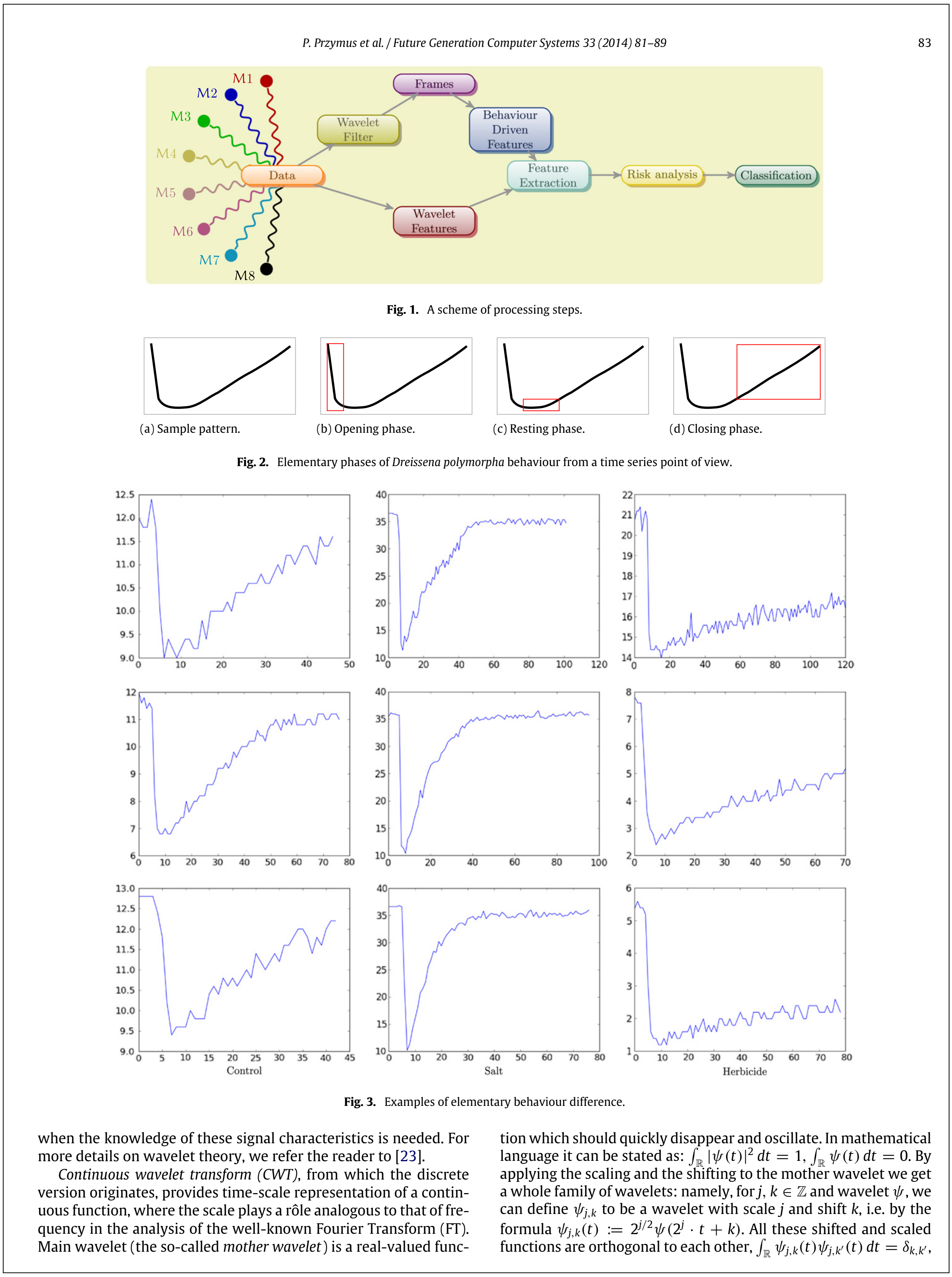
For the analysis we used the data from experiments carried out in laboratory conditions where the effect of salt and herbicides on the behaviour of mussel was examined. In the experiment we used mussels from the same colony and all experiments were conducted with at least 50% of the mussels in control conditions. In this paper, we analyse a total of 853 min of herbicide, 986 min of salt and 2193 min of control measurements.

Due to customary maximum time response of BEWS system [11], we analyse data set of length 1000 s (16 min and 40 s), which from now on will be called *fragments* (or *moving windows*, since these fragments are moving right over time with the velocity 1 time point per 1 s). In the data preparation steps we remove the white noise and average the signal using a high-pass filter (wavelet filter), which passes high frequency components and completely rejects all others.

## 4. An overview of the wavelet theory

In this section, we briefly present the basic theory of wavelets and comment on their properties that are useful in the digital signal processing.

The basic theory of wavelets was established by Haar in 1909, and then developed in the mid 1960s (cf. [22]). Wavelets are very successful as an analytical tool in the representation of signals, denoising, data compression and time-scale analysis of a time series. In many situations the results obtained by this method are better than those obtained by a Fourier analysis or other methods of filtration, because wavelet transform can be used to analyse even non-stationary signals. Moreover, the coefficients in wavelet expansion include both information about the frequency and time of the peak (*multiscale resolution analysis*). This differentiates them from the Fourier transform and makes them particularly useful



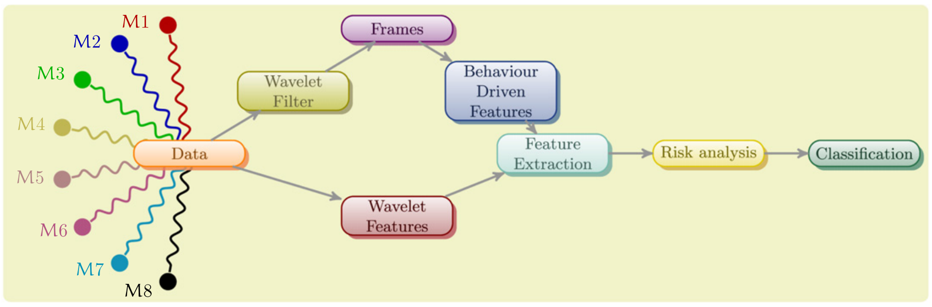
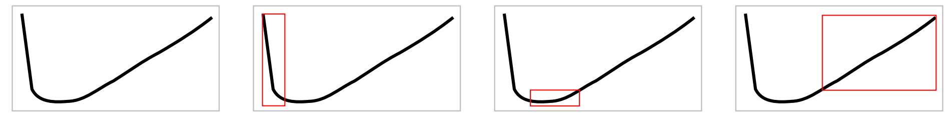


**Fig. 1.** A scheme of processing steps.



(a) Sample pattern. (b) Opening phase. (c) Resting phase. (d) Closing phase.

**Fig. 2.** Elementary phases of *Dreissena polymorpha* behaviour from a time series point of view.

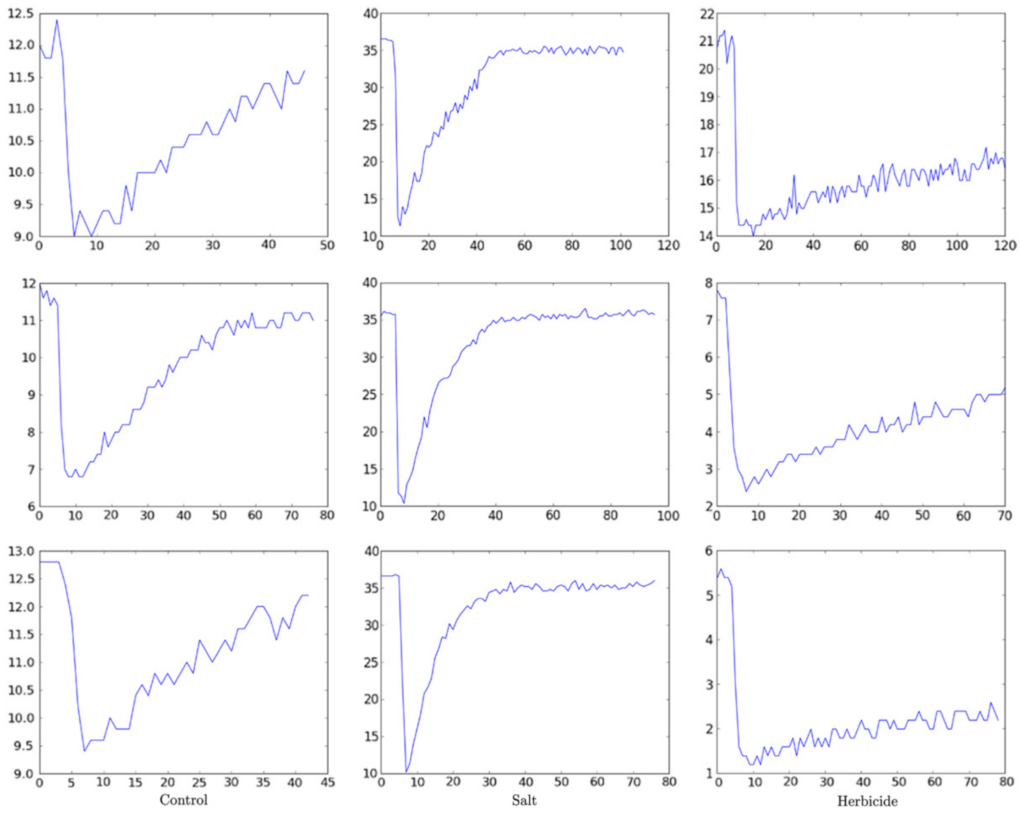
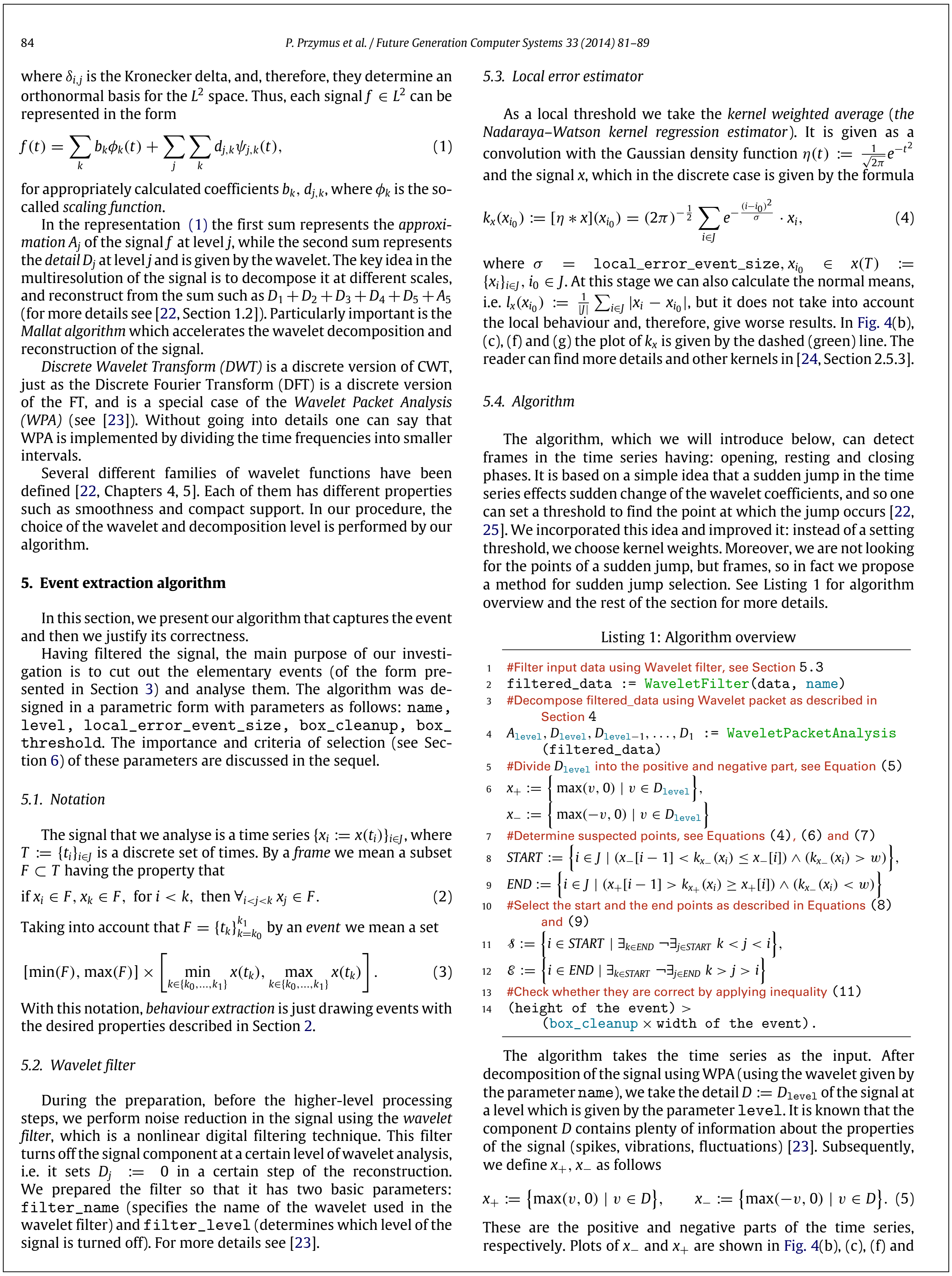


**Fig. 3.** Examples of elementary behaviour difference.

when the knowledge of these signal characteristics is needed. For more details on wavelet theory, we refer the reader to [23].

*Continuous wavelet transform (CWT)*, from which the discrete version originates, provides time-scale representation of a continuous function, where the scale plays a rôle analogous to that of frequency in the analysis of the well-known Fourier Transform (FT). Main wavelet (the so-called *mother wavelet*) is a real-valued function which should quickly disappear and oscillate. In mathematical language it can be stated as: $\int_{\mathbb{R}} |\psi(t)|^2\,dt = 1$, $\int_{\mathbb{R}} \psi(t)\,dt = 0$. By applying the scaling and the shifting to the mother wavelet we get a whole family of wavelets: namely, for $j, k \in \mathbb{Z}$ and wavelet $\psi$, we can define $\psi_{j,k}$ to be a wavelet with scale $j$ and shift $k$, i.e. by the formula $\psi_{j,k}(t) := 2^{j/2}\psi(2^j \cdot t + k)$. All these shifted and scaled functions are orthogonal to each other, $\int_{\mathbb{R}} \psi_{j,k}(t)\psi_{j,k'}(t)\,dt = \delta_{k,k'}$,



where $\delta_{i,j}$ is the Kronecker delta, and, therefore, they determine an orthonormal basis for the $L^2$ space. Thus, each signal $f \in L^2$ can be represented in the form

$$f(t) = \sum_k b_k \phi_k(t) + \sum_j \sum_k d_{j,k} \psi_{j,k}(t), \qquad (1)$$

for appropriately calculated coefficients $b_k$, $d_{j,k}$, where $\phi_k$ is the so-called *scaling function*.

In the representation (1) the first sum represents the *approximation* $A_j$ of the signal $f$ at level $j$, while the second sum represents the *detail* $D_j$ at level $j$ and is given by the wavelet. The key idea in the multiresolution of the signal is to decompose it at different scales, and reconstruct from the sum such as $D_1 + D_2 + D_3 + D_4 + D_5 + A_5$ (for more details see [22, Section 1.2]). Particularly important is the *Mallat algorithm* which accelerates the wavelet decomposition and reconstruction of the signal.

*Discrete Wavelet Transform (DWT)* is a discrete version of CWT, just as the Discrete Fourier Transform (DFT) is a discrete version of the FT, and is a special case of the *Wavelet Packet Analysis (WPA)* (see [23]). Without going into details one can say that WPA is implemented by dividing the time frequencies into smaller intervals.

Several different families of wavelet functions have been defined [22, Chapters 4, 5]. Each of them has different properties such as smoothness and compact support. In our procedure, the choice of the wavelet and decomposition level is performed by our algorithm.

## 5. Event extraction algorithm

In this section, we present our algorithm that captures the event and then we justify its correctness.

Having filtered the signal, the main purpose of our investigation is to cut out the elementary events (of the form presented in Section 3) and analyse them. The algorithm was designed in a parametric form with parameters as follows: `name`, `level`, `local_error_event_size`, `box_cleanup`, `box_threshold`. The importance and criteria of selection (see Section 6) of these parameters are discussed in the sequel.

### 5.1. Notation

The signal that we analyse is a time series $\{x_i := x(t_i)\}_{i \in J}$, where $T := \{t_i\}_{i\in J}$ is a discrete set of times. By a *frame* we mean a subset $F \subset T$ having the property that

$$\text{if } x_i \in F, x_k \in F, \text{ for } i < k, \text{ then } \forall_{i<j<k}\, x_j \in F. \qquad (2)$$

Taking into account that $F = \{t_k\}_{k=k_0}^{k_1}$ by an *event* we mean a set

$$[\min(F), \max(F)] \times \left[\min_{k\in\{k_0,\dots,k_1\}} x(t_k), \max_{k\in\{k_0,\dots,k_1\}} x(t_k)\right]. \qquad (3)$$

With this notation, *behaviour extraction* is just drawing events with the desired properties described in Section 2.

### 5.2. Wavelet filter

During the preparation, before the higher-level processing steps, we perform noise reduction in the signal using the *wavelet filter*, which is a nonlinear digital filtering technique. This filter turns off the signal component at a certain level of wavelet analysis, i.e. it sets $D_j := 0$ in a certain step of the reconstruction. We prepared the filter so that it has two basic parameters: `filter_name` (specifies the name of the wavelet used in the wavelet filter) and `filter_level` (determines which level of the signal is turned off). For more details see [23].

### 5.3. Local error estimator

As a local threshold we take the *kernel weighted average* (*the Nadaraya–Watson kernel regression estimator*). It is given as a convolution with the Gaussian density function $\eta(t) := \frac{1}{\sqrt{2\pi}} e^{-t^2}$ and the signal $x$, which in the discrete case is given by the formula

$$k_x(x_{i_0}) := [\eta * x](x_{i_0}) = (2\pi)^{-\frac{1}{2}} \sum_{i\in J} e^{-\frac{(i-i_0)^2}{\sigma}} \cdot x_i, \qquad (4)$$

where $\sigma$ = `local_error_event_size`, $x_{i_0} \in x(T) := \{x_i\}_{i\in J}$, $i_0 \in J$. At this stage we can also calculate the normal means, i.e. $l_x(x_{i_0}) := \frac{1}{|J|}\sum_{i\in J}|x_i - x_{i_0}|$, but it does not take into account the local behaviour and, therefore, give worse results. In Fig. 4(b), (c), (f) and (g) the plot of $k_x$ is given by the dashed (green) line. The reader can find more details and other kernels in [24, Section 2.5.3].

### 5.4. Algorithm

The algorithm, which we will introduce below, can detect frames in the time series having: opening, resting and closing phases. It is based on a simple idea that a sudden jump in the time series effects sudden change of the wavelet coefficients, and so one can set a threshold to find the point at which the jump occurs [22, 25]. We incorporated this idea and improved it: instead of a setting threshold, we choose kernel weights. Moreover, we are not looking for the points of a sudden jump, but frames, so in fact we propose a method for sudden jump selection. See Listing 1 for algorithm overview and the rest of the section for more details.

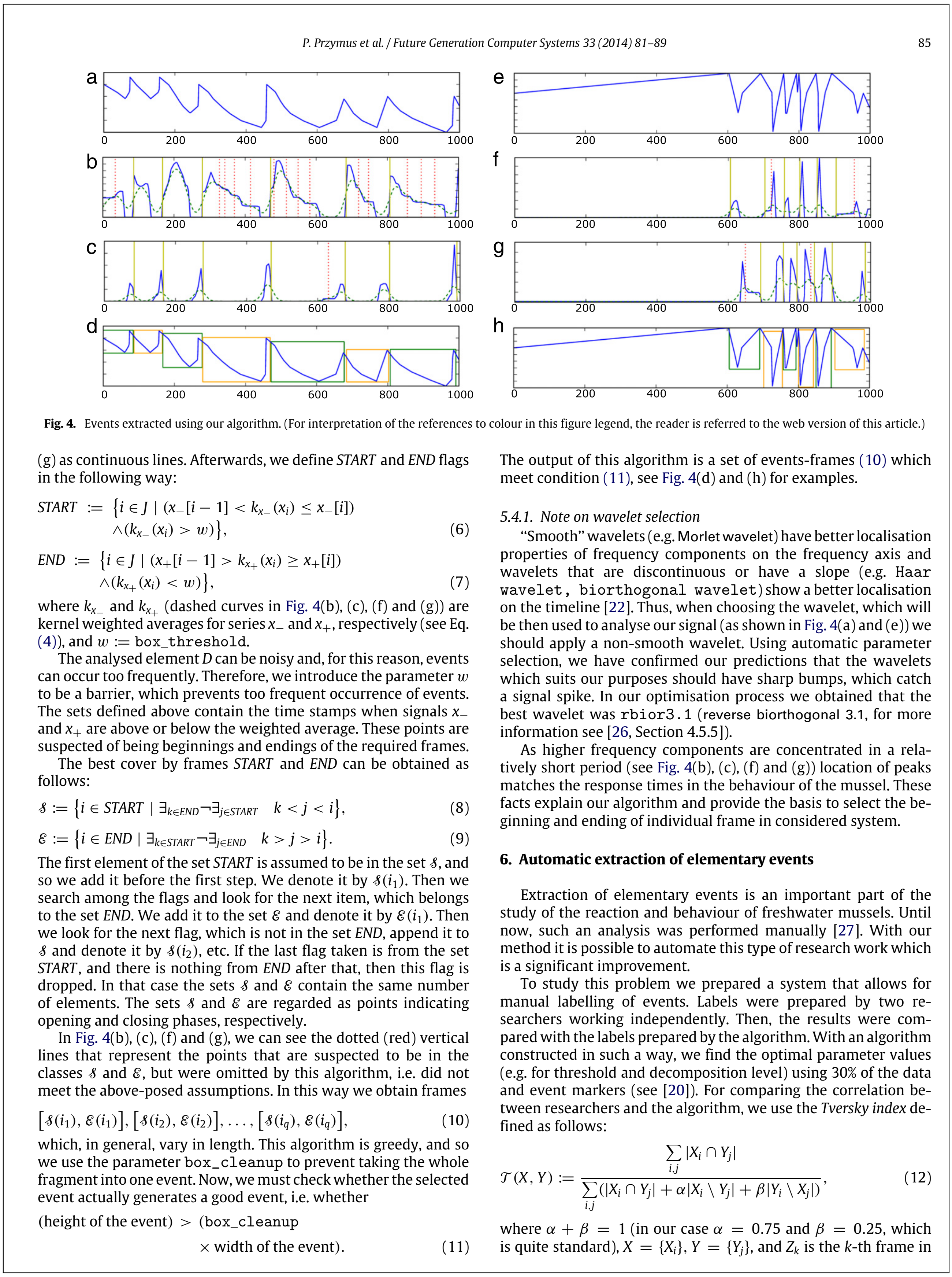
Listing 1: Algorithm overview

```
#Filter input data using Wavelet filter, see Section 5.3
filtered_data := WaveletFilter(data, name)
#Decompose filtered_data using Wavelet packet as described in
    Section 4
$A_{level}, D_{level}, D_{level-1}, \dots, D_1$ := WaveletPacketAnalysis
    (filtered_data)
#Divide $D_{level}$ into the positive and negative part, see Equation (5)
$x_+ := \{\max(v,0) \mid v \in D_{level}\}$,
$x_- := \{\max(-v,0) \mid v \in D_{level}\}$
#Determine suspected points, see Equations (4), (6) and (7)
$START := \{i \in J \mid (x_-[i-1] < k_{x_-}(x_i) \le x_-[i]) \wedge (k_{x_-}(x_i) > w)\}$,
$END := \{i \in J \mid (x_+[i-1] > k_{x_+}(x_i) \ge x_+[i]) \wedge (k_{x_-}(x_i) < w)\}$
#Select the start and the end points as described in Equations (8)
    and (9)
$\mathcal{S} := \{i \in START \mid \exists_{k\in END} \neg\exists_{j\in START}\ k < j < i\}$,
$\mathcal{E} := \{i \in END \mid \exists_{k\in START} \neg\exists_{j\in END}\ k > j > i\}$
#Check whether they are correct by applying inequality (11)
(height of the event) >
    (box_cleanup $\times$ width of the event).
```

The algorithm takes the time series as the input. After decomposition of the signal using WPA (using the wavelet given by the parameter `name`), we take the detail $D := D_{\texttt{level}}$ of the signal at a level which is given by the parameter `level`. It is known that the component $D$ contains plenty of information about the properties of the signal (spikes, vibrations, fluctuations) [23]. Subsequently, we define $x_+, x_-$ as follows

$$x_+ := \{\max(v, 0) \mid v \in D\}, \qquad x_- := \{\max(-v, 0) \mid v \in D\}. \quad (5)$$

These are the positive and negative parts of the time series, respectively. Plots of $x_-$ and $x_+$ are shown in Fig. 4(b), (c), (f) and



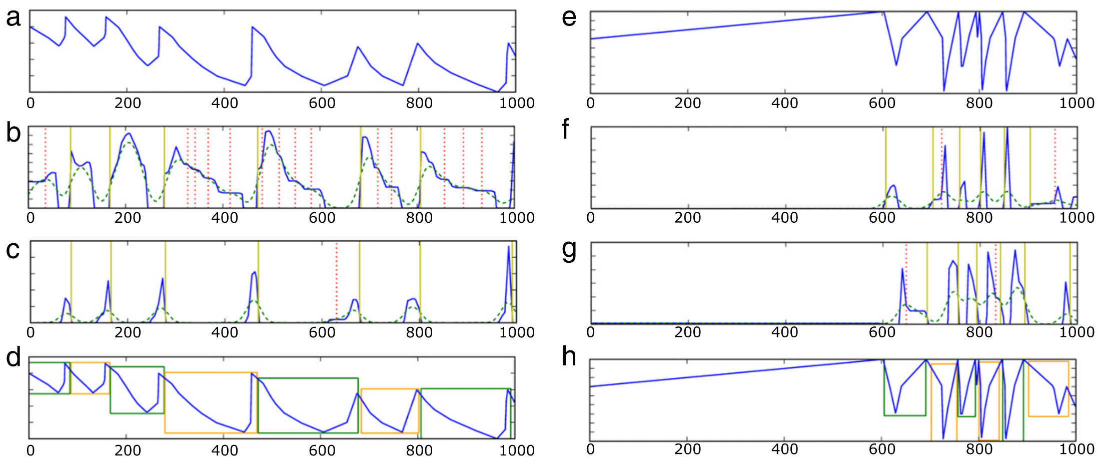


**Fig. 4.** Events extracted using our algorithm. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

(g) as continuous lines. Afterwards, we define *START* and *END* flags in the following way:

$$START := \{i \in J \mid (x_-[i-1] < k_{x_-}(x_i) \le x_-[i]) \wedge (k_{x_-}(x_i) > w)\}, \quad (6)$$

$$END := \{i \in J \mid (x_+[i-1] > k_{x_+}(x_i) \ge x_+[i]) \wedge (k_{x_+}(x_i) < w)\}, \quad (7)$$

where $k_{x_-}$ and $k_{x_+}$ (dashed curves in Fig. 4(b), (c), (f) and (g)) are kernel weighted averages for series $x_-$ and $x_+$, respectively (see Eq. (4)), and $w :=$ `box_threshold`.

The analysed element $D$ can be noisy and, for this reason, events can occur too frequently. Therefore, we introduce the parameter $w$ to be a barrier, which prevents too frequent occurrence of events. The sets defined above contain the time stamps when signals $x_-$ and $x_+$ are above or below the weighted average. These points are suspected of being beginnings and endings of the required frames.

The best cover by frames *START* and *END* can be obtained as follows:

$$\mathcal{S} := \{i \in START \mid \exists_{k \in END} \neg \exists_{j \in START} \quad k < j < i\}, \quad (8)$$

$$\mathcal{E} := \{i \in END \mid \exists_{k \in START} \neg \exists_{j \in END} \quad k > j > i\}. \quad (9)$$

The first element of the set *START* is assumed to be in the set $\mathcal{S}$, and so we add it before the first step. We denote it by $\mathcal{S}(i_1)$. Then we search among the flags and look for the next item, which belongs to the set *END*. We add it to the set $\mathcal{E}$ and denote it by $\mathcal{E}(i_1)$. Then we look for the next flag, which is not in the set *END*, append it to $\mathcal{S}$ and denote it by $\mathcal{S}(i_2)$, etc. If the last flag taken is from the set *START*, and there is nothing from *END* after that, then this flag is dropped. In that case the sets $\mathcal{S}$ and $\mathcal{E}$ contain the same number of elements. The sets $\mathcal{S}$ and $\mathcal{E}$ are regarded as points indicating opening and closing phases, respectively.

In Fig. 4(b), (c), (f) and (g), we can see the dotted (red) vertical lines that represent the points that are suspected to be in the classes $\mathcal{S}$ and $\mathcal{E}$, but were omitted by this algorithm, i.e. did not meet the above-posed assumptions. In this way we obtain frames

$$[\mathcal{S}(i_1), \mathcal{E}(i_1)], [\mathcal{S}(i_2), \mathcal{E}(i_2)], \dots, [\mathcal{S}(i_q), \mathcal{E}(i_q)], \quad (10)$$

which, in general, vary in length. This algorithm is greedy, and so we use the parameter `box_cleanup` to prevent taking the whole fragment into one event. Now, we must check whether the selected event actually generates a good event, i.e. whether

$$(\text{height of the event}) > (\texttt{box\_cleanup} \times \text{width of the event}). \quad (11)$$

The output of this algorithm is a set of events-frames (10) which meet condition (11), see Fig. 4(d) and (h) for examples.

*5.4.1. Note on wavelet selection*

"Smooth" wavelets (e.g. Morlet wavelet) have better localisation properties of frequency components on the frequency axis and wavelets that are discontinuous or have a slope (e.g. `Haar wavelet, biorthogonal wavelet`) show a better localisation on the timeline [22]. Thus, when choosing the wavelet, which will be then used to analyse our signal (as shown in Fig. 4(a) and (e)) we should apply a non-smooth wavelet. Using automatic parameter selection, we have confirmed our predictions that the wavelets which suits our purposes should have sharp bumps, which catch a signal spike. In our optimisation process we obtained that the best wavelet was `rbior3.1` (reverse biorthogonal 3.1, for more information see [26, Section 4.5.5]).

As higher frequency components are concentrated in a relatively short period (see Fig. 4(b), (c), (f) and (g)) location of peaks matches the response times in the behaviour of the mussel. These facts explain our algorithm and provide the basis to select the beginning and ending of individual frame in considered system.

## 6. Automatic extraction of elementary events

Extraction of elementary events is an important part of the study of the reaction and behaviour of freshwater mussels. Until now, such an analysis was performed manually [27]. With our method it is possible to automate this type of research work which is a significant improvement.

To study this problem we prepared a system that allows for manual labelling of events. Labels were prepared by two researchers working independently. Then, the results were compared with the labels prepared by the algorithm. With an algorithm constructed in such a way, we find the optimal parameter values (e.g. for threshold and decomposition level) using 30% of the data and event markers (see [20]). For comparing the correlation between researchers and the algorithm, we use the *Tversky index* defined as follows:

$$\mathcal{T}(X, Y) := \frac{\sum_{i,j} |X_i \cap Y_j|}{\sum_{i,j} (|X_i \cap Y_j| + \alpha |X_i \setminus Y_j| + \beta |Y_i \setminus X_j|)}, \quad (12)$$

where $\alpha + \beta = 1$ (in our case $\alpha = 0.75$ and $\beta = 0.25$, which is quite standard), $X = \{X_i\}$, $Y = \{Y_j\}$, and $Z_k$ is the $k$-th frame in

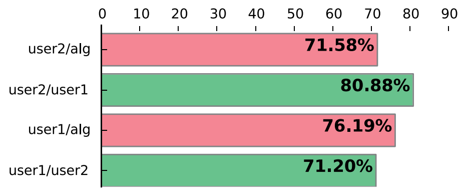


**Fig. 5.** Automatic extraction of elementary events—comparison of results.

$Z = \{Z_k\}$, a number $|Z_k|$ is its length and $Z$ is the set of frames from a given fragment. This asymmetric similarity measure compares a *variant* $X$ to a *prototype* $Y$.

Detailed results of our comparison are shown in Fig. 5. First, we made a test between the labels provided by two users to check the level of compatibility that can be achieved. Since the measure is asymmetric, their comparisons are marked as "user1/user2" and "user2/user1". Next we compared the correlation of the results obtained by the algorithm with each user. Notice that, the algorithm/user similarities are comparable with user/user similarities. Therefore, our algorithm achieved the results that are comparable to that obtained by the researchers. For this reason, the algorithm can successfully replace the manual extraction of events.

## 7. Framework for automatic classification of control and stress conditions

The second important step in our study was an attempt to automatically classify the situations when stress influences the life of mussels. The main difference between our system and other solutions (see [12,14] and the references therein) is the fact that we focus on situations that do not cause a strong stress response of mussels, however, they can pose a real threat to the environment. An example of such substances are herbicides, which despite having a high concentration in water do not cause an immediate strong reaction in mussels, but still can be dangerous for the environment in general.

Our algorithm considers the whole spectrum of behaviours since we classify activity of mussels during their normal life cycles. Stress may be so weak, so that it does not result in closures, and only a change of life cycle tells us that something has happened. This differs from the approach described in [12,7] which examines only rapid responses and omits the case described here.

We examined the usability of our algorithm for the classification. We use the same data as above, i.e. (H) herbicide, (S) salt and (C) control. We prepared three tests in which we study the efficiency of the classification:

Test 1. *The appearance of stressful substance. Risk analysis.* We create two groups, one consists only of (C), second which will be denoted by (R) simulates a group of stressful substances and consists of (H) and (S). This test checks whether we are able to correctly identify the emergence of a stressful substance (compare Fig. 6).

Test 2. *Control versus single stressful substance.* We create two pairs: (H)/(C) and (S)/(C) and then check the efficiency of classification. In this way, we investigate whether specific substances actually affect the behaviour of mussels and their appearance can be detected (see Fig. 8).

Test 3. *Identification of stressful substance.* We build a collection consisting of (H) and (S) and then classify data in order to assign them to specific groups. This allows us to analyse the possibility of identifying a particular substance one at a time when we know that there is one in the environment already (see results in Fig. 7).

### 7.1. Features set

Each method of classification needs some features from the sample. Note that we will examine fragments (moving windows) since a single event can be degenerated and only a larger set of behaviours contains adequate information. We briefly will describe the process below.

In order to classify the events we use the following simple statistical estimators (behaviour driven features):

$$\text{Behaviour driven feat.} := \Big[\mathrm{len}(\mathcal{E}_F),\ \mathrm{med}\big(X\big),\ \max\big(X\big),\ \min\big(X\big),\ \mathrm{mean}\big(X\big),\ \mathrm{std}\big(X\big)\Big], \tag{13}$$

where $X = \{\mathrm{len}(E)\}_{E\in\mathcal{E}_F}$ and $\mathcal{E}_F$ is a set of events (i.e. frames) from the fragment $F$. However, this set of characteristics is not chosen arbitrarily as we shall see below.

In addition, in order to represent the time frequency distribution we combined the above-mentioned vector of features (13) with the following features (cf. [19]): maximum, minimum, mean and standard deviation of the wavelet coefficients resulting from DWT in each event, i.e.:

$$\text{Wavelet feat.} := \Big[\Big\{\max_{E\in\mathcal{E}_F} D_i^E\Big\}_{i=1}^{5},\ \max_{E\in\mathcal{E}_F} A_5^E,\ \Big\{\min_{E\in\mathcal{E}_F} D_i^E\Big\}_{i=1}^{5},\ \min_{E\in\mathcal{E}_F} A_5^E,\ \Big\{\operatorname*{mean}_{E\in\mathcal{E}_F} D_i^E\Big\}_{i=1}^{5},\ \operatorname*{mean}_{E\in\mathcal{E}_F} A_5^E,\ \Big\{\operatorname*{std}_{E\in\mathcal{E}_F} D_i^E\Big\}_{i=1}^{5},\ \mathrm{std}_{E\in\mathcal{E}_F} A_5^E\Big], \tag{14}$$

where $D_i^E$, $A_i^E$ denotes the detail and the approximation of the event $E$ at the $i$-th level in the DWT (see Section 4). Here the level of decomposition is set to 5. In conclusion, we define:

$$\text{Combined features} := \Big[\text{Behaviour driven feat., Wavelet feat.}\Big]. \tag{15}$$

*Preliminary data processing.* To improve the effectiveness of classification algorithm, a commonly used technique is to standardise and/or normalise observations. Standardisation brings us to a situation in which the expected value of the sample is 0 and standard deviation equals 1, normalisation brings variables to the interval [0, 1]. We used the following formulae

$$\begin{aligned} \mathrm{standardise}\big(z_j[i]\big) &:= \frac{z_j[i] - \mathrm{mean}(z_j)}{\mathrm{std}(z_j)}, \\ \mathrm{normalise}\big(z_j[i]\big) &:= \frac{z_j[i] - \min(z_j)}{\max(z_j) - \min(z_j)}, \end{aligned} \tag{16}$$

where $i$ is index of the vector $z_j$, i.e. $z_j := (z_j[i])$, $j$ is the index of the feature (variable).

Due to the fact that a poorly chosen set of features may reduce the efficiency of classification algorithm, we measure the significance of the proposed features and then take the set of features that are significant. We apply the method of Iterative RELIEF to see which features were leading in each class. Iterative RELIEF is an extension of RELIEF which is an online solution to a convex optimisation problem [28]. With this tool we can iteratively estimate the optimal feature weights, so that we do not rely on manual searching (which can be subjective) and, therefore, provide an optimal solution.

Initially the set of behaviour driven features was larger: it contained few extra statistical estimators, e.g. the "peak-to-peak" (min-to-max, ptp) function, autocorrelation, variation, etc. But after using IRELIEF we were able to reduce the set of characteristics and obtained those shown in Eq. (13). It is worth to mention that using them we attained significantly better results than with the original set.

### 7.1.1. Classification methods and validation

We use the following classification methods:

1. *k-Nearest Neighbours* (*k-NN*) is a method in which we look for similar objects rather than fitting a model. Methods for predicting *k*-nearest neighbours are determined on the basis of *k* objects from the learning sample that are closed (in the sense of the distance) to the objects for which we determine the value of the dependent variable [24, Section 2.5.4].
2. *Spectral Regression Discriminant Analysis (SRDA)* is a classification method proposed recently [29]. It was developed from the well-known Linear Discriminant Analysis (LDA) and is based on graph analysis. In SRDA it is needed to solve a set of least square problems (in fact, linear equations) and no eigenvectors computation is needed (as it was common in LDA), therefore, it saves both time and memory and can be easily scaled to very large high-dimensional data sets.
3. *Fisher Discriminant Analysis (FDA)* measures, in some sense, a ratio of signal to the noise. It is sometimes confused with LDA, but it describes a slightly different discriminant. In this classification method high-dimensional data is projected on the line and then classification in one-dimensional space is performed.

*Validation procedure*. Standard approach to assess the predictive performance of the classification method is training a system and comparing it on independent data, sometimes called *validation set*. For small validation sets the traditional solution is a *cross-validation* [24]. After selecting the above-mentioned features we cross-validate classifiers in the distribution of the $K = 10$ groups to check if there is a possibility of classification. The average percentage results of validation are presented below, along with a discussion of results. All decision classes were approximately balanced.

### 7.1.2. Discussion on the classification results

It should be emphasised that in our experiments we found no abrupt reaction, however such reactions were the basis of earlier BEWS [12,15,13]. Therefore, in some situations this type of threat can likely be unnoticed.

In Test 1 (Fig. 6) we investigate possibility of determining the appearance of stressful substance. The best results are obtained using wavelet features and combination of both features, wavelets and ours; all classifiers exceeded the 70%. Behaviour driven features were not sufficient in this case, reaching only a little over 60%. The best classifiers here were based on the SRDA and the FDA (both give more than 75%).

In the case where we study the control versus single stressful substance i.e. Test 2 (Fig. 8), the best results were obtained by the combined features: (H) scores in the range 77%–81% and (S) obtained 86%–92%. The best classifier was again SRDA.

In Test 3 (Fig. 7) all sets of features proved to give similar results as in the case of *k*-NN and SRDA. The results fluctuated around 78%–83% for SRDA and 81%–83% for *k*-NN.

Note that, when the observation is constructed on the basis of all the features described in Eq. (15) we can significantly improve the efficiency of classification compared to set of attributes based only on a set of features presented in Eq. (13) or that proposed in [19].

We were able to test a proposed framework on three different aspects. The obtained results constitute a solid basis for further research into the use of zebra mussels in BEWS-like systems. The results also confirm the emergence of more complex reactions at a time when there is a threat.

The best results we obtained when we used sets of wavelet features and a combined set of features. It means that a set based on the behaviour can be a valuable addition to complex characteristics

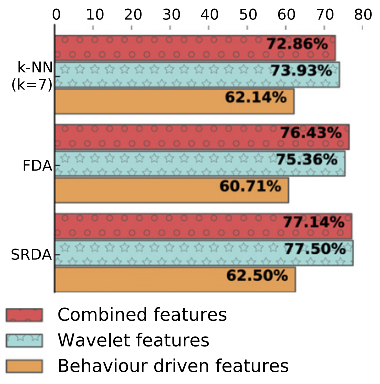


**Fig. 6.** The appearance of stressful substance.

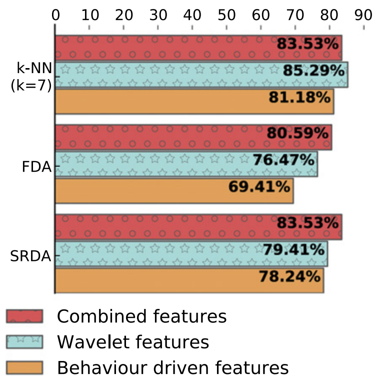


**Fig. 7.** Identification of stressful substance.

and improve results. Note that for the construction of a set of attributes, using elementary events, we used only a fraction of the features that could be used. In the future work we plan to use more characteristics that can be drawn from elementary events.

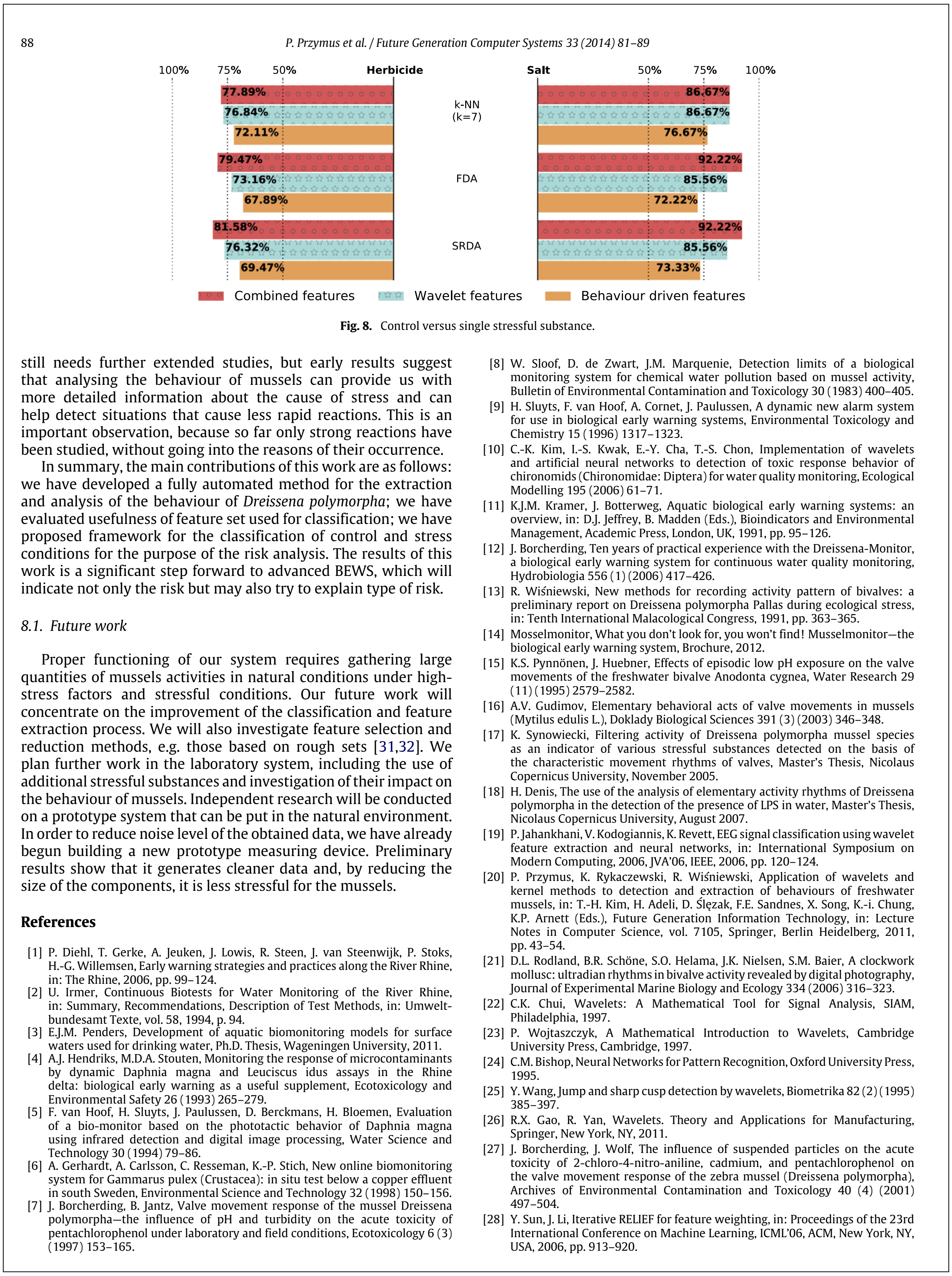
### 7.1.3. Algorithm performance

Whole process is suitable for continuous monitoring in terms of processing time. Even on low-end computer, features extraction and classification of 16 min 40 s of observation from one sensor (i.e. fragment) does not exceeds 1 s. For details on used hardware and features algorithm performance see [20, Section 5]. We have used mlpy [30] as machine learning framework (see documentation for details of implementation and performance).

## 8. Conclusions

Stressful situations can change the behaviour of mussels (both at the level of fundamental changes in the life cycles and a rhythm disturbance of these behaviours). It can also cause an emergence of a new behaviour (i.e. testing the surrounding environment).

In this paper, we presented a fast algorithm for the extraction of elementary events. We tested its usability in two practical applications: as a tool for laboratory work (automation of the extraction of elementary events can significantly accelerate research on the behaviour and reactions of mussels in different situations) and as a tool for creating a set of observations. Furthermore, we have proposed framework for the classification of control and stress conditions. By analysing our data we were able to prepare, successfully, three tests showing the effectiveness of the classification of control and stress conditions. Our solution



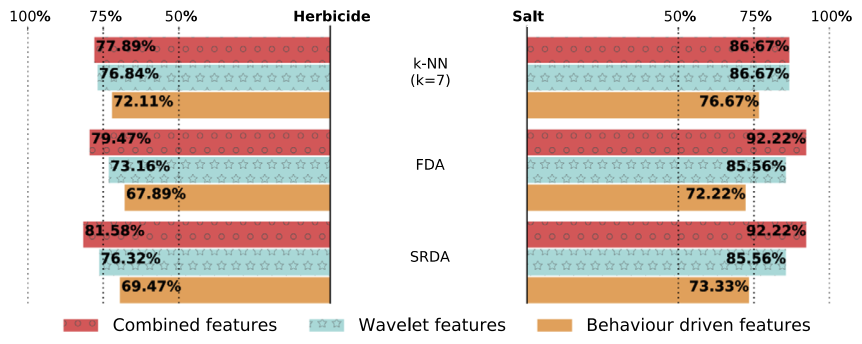


**Fig. 8.** Control versus single stressful substance.

still needs further extended studies, but early results suggest that analysing the behaviour of mussels can provide us with more detailed information about the cause of stress and can help detect situations that cause less rapid reactions. This is an important observation, because so far only strong reactions have been studied, without going into the reasons of their occurrence.

In summary, the main contributions of this work are as follows: we have developed a fully automated method for the extraction and analysis of the behaviour of *Dreissena polymorpha*; we have evaluated usefulness of feature set used for classification; we have proposed framework for the classification of control and stress conditions for the purpose of the risk analysis. The results of this work is a significant step forward to advanced BEWS, which will indicate not only the risk but may also try to explain type of risk.

### *8.1. Future work*

Proper functioning of our system requires gathering large quantities of mussels activities in natural conditions under high-stress factors and stressful conditions. Our future work will concentrate on the improvement of the classification and feature extraction process. We will also investigate feature selection and reduction methods, e.g. those based on rough sets [31,32]. We plan further work in the laboratory system, including the use of additional stressful substances and investigation of their impact on the behaviour of mussels. Independent research will be conducted on a prototype system that can be put in the natural environment. In order to reduce noise level of the obtained data, we have already begun building a new prototype measuring device. Preliminary results show that it generates cleaner data and, by reducing the size of the components, it is less stressful for the mussels.

## References

[1] P. Diehl, T. Gerke, A. Jeuken, J. Lowis, R. Steen, J. van Steenwijk, P. Stoks, H.-G. Willemsen, Early warning strategies and practices along the River Rhine, in: The Rhine, 2006, pp. 99–124.

[2] U. Irmer, Continuous Biotests for Water Monitoring of the River Rhine, in: Summary, Recommendations, Description of Test Methods, in: Umwelt-bundesamt Texte, vol. 58, 1994, p. 94.

[3] E.J.M. Penders, Development of aquatic biomonitoring models for surface waters used for drinking water, Ph.D. Thesis, Wageningen University, 2011.

[4] A.J. Hendriks, M.D.A. Stouten, Monitoring the response of microcontaminants by dynamic Daphnia magna and Leuciscus idus assays in the Rhine delta: biological early warning as a useful supplement, Ecotoxicology and Environmental Safety 26 (1993) 265–279.

[5] F. van Hoof, H. Sluyts, J. Paulussen, D. Berckmans, H. Bloemen, Evaluation of a bio-monitor based on the phototactic behavior of Daphnia magna using infrared detection and digital image processing, Water Science and Technology 30 (1994) 79–86.

[6] A. Gerhardt, A. Carlsson, C. Resseman, K.-P. Stich, New online biomonitoring system for Gammarus pulex (Crustacea): in situ test below a copper effluent in south Sweden, Environmental Science and Technology 32 (1998) 150–156.

[7] J. Borcherding, B. Jantz, Valve movement response of the mussel Dreissena polymorpha—the influence of pH and turbidity on the acute toxicity of pentachlorophenol under laboratory and field conditions, Ecotoxicology 6 (3) (1997) 153–165.

[8] W. Sloof, D. de Zwart, J.M. Marquenie, Detection limits of a biological monitoring system for chemical water pollution based on mussel activity, Bulletin of Environmental Contamination and Toxicology 30 (1983) 400–405.

[9] H. Sluyts, F. van Hoof, A. Cornet, J. Paulussen, A dynamic new alarm system for use in biological early warning systems, Environmental Toxicology and Chemistry 15 (1996) 1317–1323.

[10] C.-K. Kim, I.-S. Kwak, E.-Y. Cha, T.-S. Chon, Implementation of wavelets and artificial neural networks to detection of toxic response behavior of chironomids (Chironomidae: Diptera) for water quality monitoring, Ecological Modelling 195 (2006) 61–71.

[11] K.J.M. Kramer, J. Botterweg, Aquatic biological early warning systems: an overview, in: D.J. Jeffrey, B. Madden (Eds.), Bioindicators and Environmental Management, Academic Press, London, UK, 1991, pp. 95–126.

[12] J. Borcherding, Ten years of practical experience with the Dreissena-Monitor, a biological early warning system for continuous water quality monitoring, Hydrobiologia 556 (1) (2006) 417–426.

[13] R. Wiśniewski, New methods for recording activity pattern of bivalves: a preliminary report on Dreissena polymorpha Pallas during ecological stress, in: Tenth International Malacological Congress, 1991, pp. 363–365.

[14] Mosselmonitor, What you don't look for, you won't find! Musselmonitor—the biological early warning system, Brochure, 2012.

[15] K.S. Pynnönen, J. Huebner, Effects of episodic low pH exposure on the valve movements of the freshwater bivalve Anodonta cygnea, Water Research 29 (11) (1995) 2579–2582.

[16] A.V. Gudimov, Elementary behavioral acts of valve movements in mussels (Mytilus edulis L.), Doklady Biological Sciences 391 (3) (2003) 346–348.

[17] K. Synowiecki, Filtering activity of Dreissena polymorpha mussel species as an indicator of various stressful substances detected on the basis of the characteristic movement rhythms of valves, Master's Thesis, Nicolaus Copernicus University, November 2005.

[18] H. Denis, The use of the analysis of elementary activity rhythms of Dreissena polymorpha in the detection of the presence of LPS in water, Master's Thesis, Nicolaus Copernicus University, August 2007.

[19] P. Jahankhani, V. Kodogiannis, K. Revett, EEG signal classification using wavelet feature extraction and neural networks, in: International Symposium on Modern Computing, 2006, JVA'06, IEEE, 2006, pp. 120–124.

[20] P. Przymus, K. Rykaczewski, R. Wiśniewski, Application of wavelets and kernel methods to detection and extraction of behaviours of freshwater mussels, in: T.-H. Kim, H. Adeli, D. Ślęzak, F.E. Sandnes, X. Song, K.-i. Chung, K.P. Arnett (Eds.), Future Generation Information Technology, in: Lecture Notes in Computer Science, vol. 7105, Springer, Berlin Heidelberg, 2011, pp. 43–54.

[21] D.L. Rodland, B.R. Schöne, S.O. Helama, J.K. Nielsen, S.M. Baier, A clockwork mollusc: ultradian rhythms in bivalve activity revealed by digital photography, Journal of Experimental Marine Biology and Ecology 334 (2006) 316–323.

[22] C.K. Chui, Wavelets: A Mathematical Tool for Signal Analysis, SIAM, Philadelphia, 1997.

[23] P. Wojtaszczyk, A Mathematical Introduction to Wavelets, Cambridge University Press, Cambridge, 1997.

[24] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[25] Y. Wang, Jump and sharp cusp detection by wavelets, Biometrika 82 (2) (1995) 385–397.

[26] R.X. Gao, R. Yan, Wavelets. Theory and Applications for Manufacturing, Springer, New York, NY, 2011.

[27] J. Borcherding, J. Wolf, The influence of suspended particles on the acute toxicity of 2-chloro-4-nitro-aniline, cadmium, and pentachlorophenol on the valve movement response of the zebra mussel (Dreissena polymorpha), Archives of Environmental Contamination and Toxicology 40 (4) (2001) 497–504.

[28] Y. Sun, J. Li, Iterative RELIEF for feature weighting, in: Proceedings of the 23rd International Conference on Machine Learning, ICML'06, ACM, New York, NY, USA, 2006, pp. 913–920.

[29] D. Cai, X. He, J. Han, SRDA: an efficient algorithm for large-scale discriminant analysis, The IEEE Transactions on Knowledge and Data Engineering 20 (1) (2008) 1–12.

[30] D. Albanese, R. Visintainer, S. Merler, S. Riccadonna, G. Jurman, C. Furlanello, mlpy: Machine Learning Python, 2012.

[31] S. Widz, D. Ślęzak, Rough set based decision support—models easy to interpret, in: Rough Sets: Selected Methods and Applications in Management and Engineering, 2012, pp. 95–112.

[32] A. Wieczorkowska, J. Wróblewski, P. Synak, D. Ślęzak, Application of temporal descriptors to musical instrument sound recognition, Journal of Intelligent Information Systems 21 (1) (2003) 71–93.

**Piotr Przymus** is a Ph.D. Student of Computer Science in the Faculty of Mathematics and Computer Science at Nicolaus Copernicus University. His main scientific interests are: database systems, GPGPU computing and data mining.

**Krzysztof Rykaczewski** is a Ph.D. Student of Mathematics in the Faculty of Mathematics and Computer Science at Nicolaus Copernicus University. His research interests concern control theory and partial differential equations.

**Ryszard Wiśniewski** is the Head of the Laboratory of Applied Hydrobiology at Nicolaus Copernicus University, Torun. He has research interests in biomonitoring and water pollution.

# B APPENDIX: THE DESCRIPTION OF THE EXPERIMENTS

## B.1 REPOSITORY OF SOURCE CODE AND DATASETS

The source code of programs and datasets necessary to reproduce the results presented in this thesis can be downloaded from the Web. In order to obtain them please fill in the form on `http://phd.przymus.org` and follow the instructions sent in a response email. Alternatively, please contact me directly using this email address: `eror@mat.umk.pl`.

## B.2 ORGANIZATION OF THE REPOSITORY

The repository contains three directories:

- LightweightCompressionAndQueryProcessing — The source code for lightweight compression methods and time series query processing methods described in [58, 59, 61, 62] (see Appendix A.5 on page 69, Appendix A.1 on page 27, Appendix A.3 on page 47 and Appendix A.6 on page 81, respectively).
- HeterogeneousQueryPlans — The source code of the bi-objective query planner and the simulator described in [63] may be found in this directory. See Appendix A.4 on page 55 for an overview.
- DreisennaPolymorphaMonitoring — The source code and datasets used in experiments described in [61, 62]. Consult Appendix A.6 on page 81 and Appendix A.5 on page 69 for more details.

In each directory there is a `README.md` file which describes in details experiments contained therein, including hardware and libraries requirements, compilation process and how to execute the experiments. Please bear in mind that results of experiments depend on the hardware used (especially on the GPU card used).

## B.3 LICENSING

Please consult `LICENSE.txt` file in each directory for licensing details for source code and provided datasets.

# BIBLIOGRAPHY

[1] Business intelligence and analytics software - SAS, 2013. URL http://www.sas.com/.

[2] Marketmap analytic platform, 2013. URL http://www.sungard.com/fame.

[3] Andrzejewski, W. and Wrembel, R. GPU-WAH: Applying GPUs to compressing bitmap indexes with word aligned hybrid. In *Database and Expert Systems Applications*, pages 315–329. Springer, 2010.

[4] Australian Bureau of Statistics. Interpreting time series data, December 2001. Archived copy http://goo.gl/e9HtTJ.

[5] Barbon Jr, S., Guido, R., Vieira, L., Silva Fonseca, E., Sanchez, F., Scalassara, P., Maciel, C., Pereira, J., and Chen, S. Wavelet-based dynamic time warping. *Journal of Computational and Applied Mathematics*, 227(2):271–287, 2009.

[6] Bishop, C. M. and Nasrabadi, N. M. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[7] Bocklisch, A. Parametric fuzzy modelling framework for complex data–inherent structures. 2009.

[8] Boncz, P. *Monet – a next-Generation DBMS Kernel For Query-Intensive Applications*. 2002.

[9] Boncz, P., Zukowski, M., and Nes, N. MonetDB/X100: Hyper-pipelining query execution. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA*, pages 225–237, 2005.

[10] Breß, S. and Saake, G. Why it is time for a HyPE: A hybrid query processing engine for efficient GPU coprocessing in DBMS. *Proceedings of the VLDB Endowment*, 6(12): 1398–1403, 2013.

[11] Breß, S., Beier, F., Rauhe, H., Schallehn, E., Sattler, K.-U., and Saake, G. Automatic selection of processing units for coprocessing in databases. In *Advances in Databases and Information Systems*, pages 57–70. Springer, 2012.

[12] Breß, S., Mohammad, S., and Schallehn, E. Self-tuning distribution of DB-operations on hybrid CPU/GPU platforms. *Grundlagen von Datenbanken, CEUR-WS*, pages 89–94, 2012.

[13] Breß, S., Geist, I., Schallehn, E., Mory, M., and Saake, G. A framework for cost based optimization of hybrid CPU/GPU query plans in database systems. *Control and Cybernetics*, pages 27–35, 2013.

[14] Breß, S., Heimel, M., Siegmund, N., Bellatreche, L., and Saake, G. Exploring the design space of a GPU-aware database architecture. *New Trends in Databases and Information Systems*, pages 225–234, 2013.

[15] Breß, S., Schallehn, E., and Geist, I. Towards optimization of hybrid CPU/GPU query plans in database systems. In *New Trends in Databases and Information Systems*, pages 27–35. Springer, 2013.

[16] Buyya, R., Abramson, D., Giddy, J., and Stockinger, H. Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542, 2002.

[17] Chen, Z., Gehrke, J., and Korn, F. Query optimization in compressed database systems. In *ACM SIGMOD Record*, volume 30, pages 271–282. ACM, 2001.

[18] Chui, C. K. *Wavelets: A mathematical tool for signal analysis*. SIAM, Philadelphia, 1997.

[19] Cloudkick. 4 Months with Cassandra, a love story, March 2010. URL `https://www.cloudkick.com/blog/2010/mar/02/4_months_with_cassandra/`. Archived copy `http://goo.gl/4rw3sW`.

[20] Delbru, R., Campinas, S., Samp, K., and Tummarello, G. Adaptive frame of reference for compressing inverted lists. 2010.

[21] Deri, L., Mainardi, S., and Fusco, F. TSDB: A compressed database for time series. In *Traffic Monitoring and Analysis*, pages 143–156. Springer, 2012.

[22] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[23] Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern classification*. John Wiley & Sons, 2012.

[24] Esling, P. and Agon, C. Time-series data mining. *ACM Computing Surveys (CSUR)*, 45 (1):12, 2012.

[25] Fang, W., He, B., and Luo, Q. Database compression on graphics processors. *Proceedings of the VLDB Endowment*, 3(1-2):670–680, 2010.

[26] Fink, E. and Gandhi, H. S. Compression of time series by extracting major extrema. *Journal of Experimental and Theoretical Artificial Intelligence*, 23(2):255–270, 2011.

[27] Florescu, D. and Kossmann, D. Rethinking cost and performance of database systems. *ACM Sigmod Record*, 38(1):43–48, 2009.

[28] Franklin, M. J., Jónsson, B. T., and Kossmann, D. Performance tradeoffs for client-server query processing. In *ACM SIGMOD Record*, volume 25, pages 149–160. ACM, 1996.

[29] Garcia, V., Debreuve, E., Nielsen, F., and Barlaud, M. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 3757–3760. IEEE, 2010.

[30] Garcia, V., Debreuve, E., and Barlaud, M. Fast k nearest neighbor search using GPU. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–6. IEEE, 2008.

[31] Han, D. and Stroulia, E. A three-dimensional data model in HBase for large time-series dataset analysis. In *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the*, pages 47–56. IEEE, 2012.

[32] Han, W., Lee, J., Moon, Y., and Jiang, H. Ranked subsequence matching in time-series databases. In *Proceedings of the 33rd international conference on Very large data bases*, pages 423–434. VLDB Endowment, 2007.

[33] Herbst, G. and Bocklisch, S. Online recognition of fuzzy time series patterns. In *2009 International Fuzzy Systems Association World Congress and 2009 European Society for Fuzzy Logic and Technology Conference (IFSA-EUSFLAT 2009)*, pages 974–979, 2009.

[34] IBM. *IBM Informix TimeSeries Data User's Guide*. IBM Corporation, 2012.

[35] Jahankhani, P., Kodogiannis, V., and Revett, K. EEG signal classification using wavelet feature extraction and neural networks. In *Modern Computing, 2006. JVA'06. IEEE John Vincent Atanasoff 2006 International Symposium on*, pages 120–124. IEEE, 2006.

[36] Janosek, L. and Nemec, M. Fast polynomial approximation acceleration on the GPU. In *ICDS 2012 : The Sixth International Conference on Digital Society*, 2012.

[37] kairosdb. Fast scalable time series database, 2013. URL `https://code.google.com/p/kairosdb/`.

[38] Karimi, K., Dickson, N. G., and Hamze, F. A performance comparison of CUDA and OpenCL. *arXiv preprint arXiv:1005.2581*, 2010.

[39] Keogh, E. and Pazzani, M. Scaling up dynamic time warping to massive datasets. *Principles of Data Mining and Knowledge Discovery*, pages 1–11, 1999.

[40] Keogh, E. and Ratanamahatana, C. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[41] Keogh, E., Lonardi, S., and Chiu, B. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 550–556. ACM, 2002.

[42] Keogh, E. and Pazzani, M. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM, 2000.

[43] Keogh, E. and Pazzani, M. Derivative dynamic time warping. In *First SIAM international conference on data mining*, 2001.

[44] Krishnan, M., Neophytou, C., and Prescott, G. Wavelet transform speech recognition using vector quantization, dynamic time warping and artificial neural networks. *Center for Excellence in Computer Aided Systems Engineering and Telecommunications & Information Science Laboratory*, 1994.

[45] Last, M., Kandel, A., and Bunke, H. *Data mining in time series databases*, volume 57. World Scientific Pub Co Inc, 2004.

[46] Lees, M., Ellen, R., Steffens, M., Brodie, P., Mareels, I., and Evans, R. Information infrastructures for utilities management in the brewing industry. In *On the Move to Meaningful Internet Systems: OTM 2012 Workshops*, pages 73–77. Springer, 2012.

[47] Lin, J., Keogh, E., Lonardi, S., and Chiu, B. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM, 2003.

[48] Marler, R. T. and Arora, J. S. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[49] Mercurio, A., Di Giorgio, A., and Cioci, P. Open-source implementation of monitoring and controlling services for EMS/SCADA systems by means of web services IEC 61850 and IEC 61970 standards. *Power Delivery, IEEE Transactions on*, 24(3):1148–1153, 2009.

[50] Meyer, M. *RIAK Handbook*. Amazon Digital Services, Inc., 2011.

[51] Najafi, M., Sadoghi, M., and Jacobsen, H.-A. Flexible query processor on FPGAs. *Proceedings of the VLDB Endowment*, 6(12):1310–1313, 2013.

[52] Nvidia, C. Nvidia CUDA C programming guide. *NVIDIA Corporation*, 2011.

[53] Nvidia, C. CUDA C best practices guide version 4.1. Technical report, Tech. rep., NVIDIA Corporation, 2012.

[54] Oetiker, T. RRDtool, 2005. URL `http://people.ee.ethz.ch/oetiker/webtool/rrdtool`.

[55] OpenTSDB. What's OpenTSDB, 2010-2014. URL `http://opentsdb.net/`.

[56] Papadimitriou, C. H. and Yannakakis, M. Multiobjective query optimization. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 52–59. ACM, 2001.

[57] Poli, G., Levada, A., Mari, J., and Saito, J. Voice command recognition with dynamic time warping (dtw) using graphics processing units (gpu) with compute unified device architecture (cuda). In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, pages 19–25. IEEE, 2007.

[58] Przymus, P. and Kaczmarski, K. Improving efficiency of data intensive applications on GPU using lightweight compression. In *On the Move to Meaningful Internet Systems: OTM 2012 Workshops, Lecture Notes in Computer Science*, volume 7567, pages 3–12. Springer Berlin Heidelberg, 2012.

[59] Przymus, P. and Kaczmarski, K. Time series queries processing with GPU support. In *New Trends in Databases and Information Systems*. 17th East-European Conference on Advances in Databases and Information Systems September 1–4, 2013, Genoa, Italy, 2013.

[60] Przymus, P. and Kaczmarski, K. Dynamic compression strategy for time series database using GPU. In *New Trends in Databases and Information Systems*. 17th East-European Conference on Advances in Databases and Information Systems September 1–4, 2013, Genoa, Italy, 2013.

[61] Przymus, P., Rykaczewski, K., and Wiśniewski, R. Application of wavelets and kernel methods to detection and extraction of behaviours of freshwater mussels. *Future Generation Information Technology*, pages 43–54, 2011.

[62] Przymus, P., Rykaczewski, K., and Wiśniewski, R. Zebra mussels' behaviour detection, extraction and classification using wavelets and kernel methods. *Future Generation Computer Systems*, 33(0):81–89, 2014. Special Section on Applications of Intelligent Data and Knowledge Processing Technologies. Guest Editor: Dominik Ślęzak.

[63] Przymus, P., Kaczmarski, K., and Stencel, K. A bi-objective optimization framework for heterogeneous CPU/GPU query plans. In *CS&P 2013 Concurrency, Specification and Programming*. Proceedings of the 22nd International Workshop on Concurrency, Specification and Programming, September 25–27, 2013, Warsaw, Poland.

[64] Ratanamahatana, C. and Keogh, E. Three myths about dynamic time warping data mining. In *Proceedings of SIAM International Conference on Data Mining (SDM'05)*, pages 506–510, 2005.

[65] Ratanamahatana, C., Lin, J., Gunopulos, D., Keogh, E., Vlachos, M., and Das, G. Mining time series data. *Data Mining and Knowledge Discovery Handbook*, pages 1049–1077, 2010.

[66] Sanchez, F., Júnior, S., Guido, R., Vieira, L., and de Carvalho, N. Wavelet-based dynamic time warping for speech recognition. 2006.

[67] Sanders, J. and Kandrot, E. *CUDA by example: An introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[68] Sart, D., Mueen, A., Najjar, W., Keogh, E., and Niennattrakul, V. Accelerating dynamic time warping subsequence search with GPUs and FPGAs. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 1001–1006. IEEE, 2010.

[69] Shasha, D. Time series in finance: The array database approach, 1998. URL http://cs.nyu.edu/shasha/papers/jagtalk.html.

[70] Sonmez, O. O. and Gursoy, A. Comparison of pricing policies for a computational grid market. In *Parallel Processing and Applied Mathematics*, pages 766–773. Springer, 2006.

[71] Sprent, P. and Smeeton, N. C. *Applied nonparametric statistical methods*. CRC Press, 2010.

[72] Stonebraker, M., Aoki, P. M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., and Yu, A. Mariposa: A wide-area distributed database system. *The VLDB Journal*, 5(1): 48–63, 1996.

[73] Team R, D. C. et al. R: A language and environment for statistical computing, 2005.

[74] TempoDB. Hosted time series database service, 2013. URL https://tempo-db.com/.

[75] Teradata. Enterprise-class R for massively parallel analytics in Teradata, 9 2013. URL http://www.revolutionanalytics.com//sites/default/files/teradata_revolution_analytics_white_paper.pdf.

[76] Thawonmas, R. and Iizuka, K. Haar wavelets for online-game player classification with dynamic time warping. *Journal of Advanced Computational Intelligence Vol*, 12(2), 2007.

[77] Uguz, H., Arslan, A., Saraçoglu, R., and Türkoglu, I. Detection of heart valve diseases by using fuzzy discrete hidden Markov model. *Expert Syst. Appl.*, 34(4):2799–2811, 2008.

[78] Walkowiak, S., Wawruch, K., Nowotka, M., Ligowski, L., and Rudnicki, W. Exploring utilisation of GPU for database applications. *Procedia Computer Science*, 1(1):505–513, 2010.

[79] Wong, T. T. and Heng, P. A. Discrete wavelet transform on GPU. 2004.

[80] Wu, L., Storus, M., and Cross, D. CS315A: Final project CUDA WUDA SHUDA: CUDA compression project. Technical report, Stanford University, March 2009.

[81] Yagley, T. M. Optimizing relational databases for time series data (time series database overview part 1-3), 2013. URL http://blog.tempo-db.com/post/48645898017/optimizing-relational-databases-for-time-series-data.

[82] Zukowski, M., Boncz, P., Nes, N., and Héman, S. MonetDB/X100–a DBMS in the CPU cache. *IEEE Data Eng. Bull*, 28(2):17–22, 2005.

[83] Zukowski, M., Heman, S., Nes, N., and Boncz, P. Super-scalar RAM-CPU cache compression. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 59–59. IEEE, 2006.

## COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

`http://code.google.com/p/classicthesis/`